AFRL-AFOSR-VA-TR-2015-0316

Robust Coordination of Autonomous Systems through Risk-sensitive, Model-based Programming and Execution

Brian Williams
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

10/09/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
30–09–2015	Final	01 September 2012 — 31 August 2015

4. TITLE AND SUBTITLE

Robust Coordination of Autonomous Systems through Risk-sensitive, Model-based Programming and Execution

5a. CONTRACT NUMBER

FA9550–12–1–0348

5b. GRANT NUMBER

6926079

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Santana, Pedro; Fang, Cheng; Timmons, Eric; and Williams, Brian

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Model-based Embedded and Robotic Systems Group
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139–4301

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research (AFOSR)
801 N Randolph St., Rm. 732, Arlington VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the authors and should not be construed as an official U.S. Government position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

Unlike their human counterparts, most autonomous systems to date are not effective at characterizing or bounding mission risk. In this project, we enabled the development of risk-sensitive autonomous systems through three main contributions: first, we introduced cRMPL, an extension of RMPL where one can specify acceptable risk levels for different mission segments through the addition of chance constraints. Second, we extended the continuous planner, used by our executive, to generate and adapt plans that maximize expected utility within the risk bounds specified by the operators. Planning is performed through novel stochastic optimization algorithms that allocate user-specified risk to actions and constraints according to the benefit received. We evaluated the generality of this risk-sensitive paradigm in simulation and hardware, for autonomous air or space vehicles and humanoid logistics support robots. Benefits include increased number and complexity of vehicle missions for a fixed operational cost, increased robot safety around humans; a reduction in unacceptable mission failure or robot loss, and improved mission return within defined risk levels.

15. SUBJECT TERMS

keywords; associated words; other words

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Prof. Brian C. Williams
U	U	U	UU	68	19b. TELEPHONE NUMBER *(include area code)* (617) 253–3447

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18

Robust Coordination of Autonomous Systems through Risk-sensitive, Model-based Programming and Execution

Pedro Santana, Cheng Fang, Eric Timmons, and Brian Williams

Model-based Embedded and Robotic Systems Group
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139–4301

September 2015

Abstract

Unlike their human counterparts, most autonomous systems to date are not effective at characterizing or bounding mission risk. In this project, we enabled the development of risk-sensitive autonomous systems through three main contributions: first, we introduced cRMPL, an extension of RMPL where one can specify acceptable risk levels for different mission segments through the addition of chance constraints. Second, we extended the continuous planner, used by our executive, to generate and adapt plans that maximize expected utility within the risk bounds specified by the operators. Planning is performed through novel stochastic optimization algorithms that allocate user-specified risk to actions and constraints according to the benefit received. We evaluated the generality of this risk-sensitive paradigm in simulation and hardware, for autonomous air or space vehicles and humanoid logistics support robots. Benefits include increased number and complexity of vehicle missions for a fixed operational cost, increased robot safety around humans; a reduction in unacceptable mission failure or robot loss, and improved mission return within defined risk levels.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
30–09–2015	Final	01 September 2012 — 31 August 2015

4. TITLE AND SUBTITLE

Robust Coordination of Autonomous Systems through Risk-sensitive, Model-based Programming and Execution

5a. CONTRACT NUMBER

FA9550–12–1–0348

5b. GRANT NUMBER

6926079

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Santana, Pedro; Fang, Cheng; Timmons, Eric; and Williams, Brian

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Model-based Embedded and Robotic Systems Group
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139–4301

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research (AFOSR)
801 N Randolph St., Rm. 732, Arlington VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the authors and should not be construed as an official U.S. Government position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

Unlike their human counterparts, most autonomous systems to date are not effective at characterizing or bounding mission risk. In this project, we enabled the development of risk-sensitive autonomous systems through three main contributions: first, we introduced cRMPL, an extension of RMPL where one can specify acceptable risk levels for different mission segments through the addition of chance constraints. Second, we extended the continuous planner, used by our executive, to generate and adapt plans that maximize expected utility within the risk bounds specified by the operators. Planning is performed through novel stochastic optimization algorithms that allocate user-specified risk to actions and constraints according to the benefit received. We evaluated the generality of this risk-sensitive paradigm in simulation and hardware, for autonomous air or space vehicles and humanoid logistics support robots. Benefits include increased number and complexity of vehicle missions for a fixed operational cost, increased robot safety around humans; a reduction in unacceptable mission failure or robot loss, and improved mission return within defined risk levels.

15. SUBJECT TERMS

keywords; associated words; other words

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Prof. Brian C. Williams
U	U	U	UU	68	19b. TELEPHONE NUMBER *(include area code)* (617) 253–3447

Contents

Chapter 1

Introduction

1.1 Motivation

The trend in robotics is to transition from the traditional work-cell model, in which robots work separate from human workers in highly controlled environments, to a model in which the workspace is semi-structured, and humans and robots work within the same space to perform tasks. Drivers for this transition include the ability to operate robots with less set up, and the ability to perform tasks that are best achieved by leveraging the complementary skills of humans and robots.

To enable humans and robots to work together safely and effectively, while completing tasks under stringent temporal and safety requirements, we require robots to posses a keen sensitivity and responsiveness to the uncertainty in their environment. Current practice for ensuring task correctness and safety often requires groups of engineers to reason over a very large number of potential decisions and scenarios that might unfold during execution. Then, they manually generate fault monitoring codes and contingency procedures that account for the most likely scenarios, which is a challenging, time-consuming, and error-prone process.

The number of possible execution scenarios in unstructured environments is often overwhelming. For that reason, robot operators often respond by choosing to employ conservative, very predictable robot task execution strategies. This is particularly true when robots are tasked with safety-critical missions, such as information gathering in hostile environments (outer space, deep sea, war zones, etc.) and operation in close proximity to humans. These conservative strategies tend to be far from ideal in terms of accuracy and throughput, and are often brittle to dis-

turbances due to the execution strategy's inability to adapt to the environment. For instance, a common "safe" execution strategy consists of a precomputed sequence of actions that are robust to the worst-case scenario. Such a scenario typically requires humans and robots to work with complete separation, so as to limit the impact of uncertain human behavior on robot actions.

The Artificial Intelligence community has been developing robotic task execution strategies that improve robustness using on-line executives. These executives choose actions based on the state of the world, and adapt to temporal delays by performing dynamic scheduling. System managers, however, may resist the adoption of these more dynamic methods, due to a lack of explicit guarantees of correctness in terms of risk of task failure. In light of the latter, this project focused on the development of formal tools and algorithms allowing non-experts to easily specify desired safe behavior at a natural level of abstraction; verify the correctness of execution strategies and schedules in the presence of uncertainty; and automatically generate such safe execution strategies and schedules from a model description.

1.2 Desiderata

This project developed formal tools and algorithms that enable robots to execute tasks while achieving high performance, defined by some measure of utility. In addition to utility, these executives focus on enabling robots to achieve these tasks within deadlines specified as temporal constraints, and enable robots to perform these tasks while ensuring that hard safety guarantees are met. Achieving these goals presents a number of challenges:

1. The robot should appropriately represent and reason about environment and action uncertainty, and should use these models to predict the chances of success of a plan of action. It should also be able to plan actions optimally, while guaranteeing a probability of success specified by the user.

2. In order to be effective, the robot must be able to react to disturbances quickly, in real time, so that compensating actions (if any exist) are not delayed.

3. In order to manage environmental uncertainty, a robot must intelligently combine sensing actions with state-changing actions, so that the state-changing actions can be accomplished with an adequate probability of success.

We address these challenges using a model-based executive with three key capabilities. To address the first challenge (guaranteeing success probability), we leverage our prior work on Probabilistic Temporal Plan Networks (pTPN) [1], which extend Temporal Plan Networks with Uncertainty [2] by considering probabilistic models for uncontrollable choices and allowing chance constraints to be imposed on the violation of temporal constraints. We extend the chance-constrained approach of [3] to high-level temporal activity planning with uncertainty and sensing actions, and improve the robustness of our schedules by modeling systems of temporal constraints as Probabilistic Simple Temporal Networks [4, 5], therefore allowing activity scheduling to explicitly take into account the stochastic nature of the duration of various tasks.

To address the second challenge (achieving fast robustness to disturbances), our planner generates pTPNs with sufficient choice nodes to provide flexibility to anticipated uncontrollable events. The generated pTPN effectively encodes an optimal control policy, in the form of a conditional plan, for the correct responses to any combination of uncontrollable event outcomes. This allows the runtime executive to interpret the plan (the pTPN) to quickly make optimal decisions, no matter how the uncontrollable events turn out.

To address the third challenge (achieving adequate situation awareness), our planner generates pTPNs that optimally combine sensing and state changing actions so that the overall situation awareness is adequate to achieve the goals, within the success probability specified by the user. We leverage previously developed planning capabilities, augmenting them with the capability of combining sensing and state changing actions.

The goal of the project was to conduct basic research to create a prototype of a deployable risk-sensitive intelligent system. While our theoretical contributions were mission-enabling by allowing reasoning over probabilistic uncertainty, we also ensured that the advances were translated to useful technology. We present the high-level description of the desired characteristics which guided our efforts as follows:

1. The system *shall* be risk-sensitive. Given the environment model and a description of the actions and observations, the system will generate a plan and schedule to meet all specifications with a guarantee on the probability of success.

2. The system *shall* be scalable. The system will solve for a plan and schedule within a time frame appropriate to the number of variables in the input mission description.

3. The system *shall* have contingency plans. The system will solve for a nominal plan, but also have back up strategies in the case of particularly large deviations from what was expected.

4. The system *shall* be transferable across problem domains. The intelligent system shall allow both operations level planning and support direct control of hardware assets in a variety of problem scenarios, dealing with different vehicles across different environments.

5. The system *shall* be easy to operate by non-expert users. The intelligent system should allow the user to specify a set of desired outcomes and reason over automatically instantiated actions. As the actions are no longer hand-coded, it addresses the scalability issues which arise when describing problems with a large number of actions, and does not require the user to have prior knowledge of how each action affects the environment.

1.3 Contributions

An overview of *Enterprise*, the model-based programming and execution architecture implementing our risk-sensitive intelligent system, is given in Chapter 2. We describe the flow of information and control, from high-level planning and scheduling to low-level dispatch and execution layers, and provide evidence of *Enterprise*'s ease-of-use by non-experts, mission-enabling features, and transferability across domains by detailing how it has been deployed in support of two real-world applications: an undergraduate course at MIT involving autonomous quadcopters; and coordinating science-gathering missions for autonomous underwater vehicles operated by the Woods Hole Oceanographic Institution (WHOI). Subsequent chapters present the different modules and tools comprising *Enterprise*.

In Chapter 3, we describe cRMPL, an extension of the model-based programming language RMPL [6] that allows missions with state and temporal uncertainty, in addition to risk bounds in the form of chance constraints, to be specified at a high level of abstraction. In order to be practically useful, a model-based programming language must allow easy and reusable modeling of components and their hierarchical compositions into more complex systems; and enable programmers to specify the desired behavior of autonomous systems in terms of desired state, rather than low-level control commands. Our experiments in Chapter 7 indicate how easily cRMPL can be used as a modeling and control specification tool

by demonstrating it in the temporal coordination of science agents operating under temporal uncertainty, and in specifying an on-line execution policy that allows a robot to adapt to its human coworker in a collaborative manufacturing application.

Chapters 4 and 6 present an overview of formal verification tools that allow conditional plans with sensing and temporal uncertainty to be checked against safety specifications in the form of chance constraints. The algorithms in Chapter 4, whose detailed description is given in [1], pursue a diagnostic approach to quickly detect risky plan branches in a plan and verify if the probability of failure they incur is compatible with the given risk bounds. *Picard*, the scheduling algorithm presented in Chapter 6 and thoroughly explained in [7], is capable of handling probabilistic uncertainty in the timing of actions and compute activity schedules that are guaranteed to meet their temporal deadlines with high probability.

In situations where the manual specification of cRMPL programs is tedious or even intractably large to be done explicitly, one can use RAO*, a planning algorithm capable of deriving contingent execution policies with guarantees on the probability of success in domains with state uncertainty. It is briefly described in Chapter 5, with a scalability analysis and more detailed explanation given at the appendices.

In addition to the experimental results described in Chapter 2 and available at the scientific publications resulting from this project, we present in Chapter 7 a suite of different demonstrations evaluating *Enterprise* against the desiderata set out above. We provide evidence that our methods are generally applicable and transferable by presenting experiments in three different domains: coordination of Mars rovers; collaborative human-robot manufacturing; and unmanned aerial scouts. We also provide insight and references to research in learning probabilistic models of the environment in support of model-based reasoning and execution, such as our algorithm for learning Probabilistic Hybrid Automata (PHA) from experimental data [8]. Finally, we present our conclusions in Chapter 8.

1.4 Publications attributed to this project

For ease of reference, the following is a list of publications concerning work developed under this project:

- Fang et al., "Chance-Constrained Probabilistic Simple Temporal Problems" [7];

- Santana & Williams,"Chance-Constrained Consistency for Probabilistic Temporal Plan Networks" [1]
- Santana et al., "Learning Hybrid Models with Guarded Transitions" [8];
- Santana & Williams, "Dynamic Execution of Temporal Plans with Sensing Actions and Bounded Risk" [9];
- Santana et al., "RAO*: an Algorithm for Chance-Constrained POMDP's" (Appendix A).

Chapter 2

Risk-sensitive model-based execution

In this chapter we present an overview of *Enterprise*, the model-based programming and execution architecture used by our risk-sensitive intelligent system. We describe the flow of information and control, from high-level planning and scheduling to low-level dispatch and execution layers, and provide evidence of *Enterprise*'s ease-of-use by non-experts, mission-enabling features, and transferability across domains by detailing how it has been deployed in support of two real-world applications: an undergraduate course at MIT involving autonomous quadcopters; and coordinating science-gathering missions for autonomous underwater vehicles operated by the Woods Hole Oceanographic Institution (WHOI).

2.1 Enterprise

Modern robotic systems consist of a variety of components acting together. Broadly speaking, there is the hardware layer consisting of the actuators and sensors, the control system layer that drives the hardware to a desired configuration, the planning and execution layer that generates action sequences, and the input layer which can be a human or another automated system that provides goals. *Enterprise* is a framework for the planning and execution layer and is developed by the MERS group.

As shown in Figure 2.1, *Enterprise* accepts as input the goals that should be achieved, a model of the robot, and a model of the environment and, over time, outputs commands that are executable by the controls layer. *Enterprise* accepts

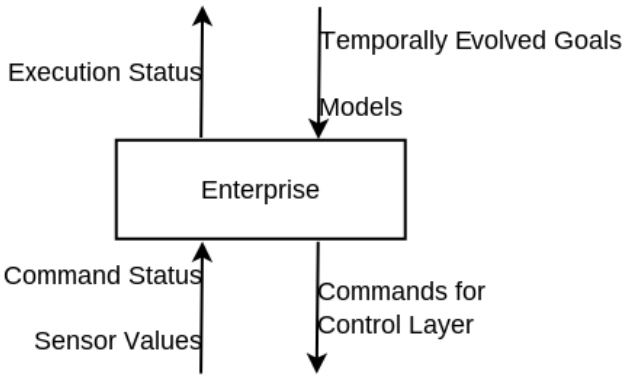


Figure 2.1: Conceptual interface of *Enterprise*.

the model and goal description in the form of an RMPL (Reactive Model-based Programming Language) program. Then, activity planners, path planners, and schedulers are called as needed to create an executable plan in the form of a temporal plan network (TPN). Last, a dispatcher is provided the executable TPN and it invokes the appropriate actions in the control layer.

Enterprise is designed for a variety of planners to be plugged into it to provide its described end-to-end capability. The current version of *Enterprise* allows only for a manually specified, static configuration of planners. Communication between planners is done through temporal plan networks, with each planner and dispatcher accepting TPNs with a subset of the allowable features of a TPN (e.g., goals, controllable decisions, uncontrollable decisions, etc.). For this work, *Enterprise* was configured to use pKirk, Picard, and Pike as shown in Figure 2.2. pKirk takes a goal description and models in the form of cRMPL and generates an executable TPN with choice. pKirk is described in depth in Chapter 5. Picard is given a chance-constrained probabilistic simple temporal problem (cc-pSTP) and determines if it is feasible. Picard is used by Kirk to check TPNs as they are being constructed and is described in depth in Chapter 6. Pike is a dispatcher and execution monitor that sends commands to the control layer to be executed at the appropriate times and monitors the sensors to know which choices in the executable TPN have been made. Pike is described in depth in [10].

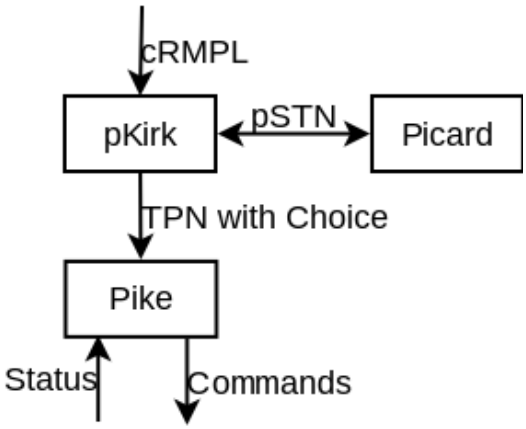


Figure 2.2: Configuration of pKirk, Picard, and Pike within *Enterprise*.

2.2 *Enterprise* deployments

The *Enterprise* system has been used in two real-world deployments by people outside of the MERS group. While neither of these deployments used specifically the two main algorithms developed in this project, pKirk or Picard, both detailed later in this report, they serve to demonstrate the broad applicability and transferability across domains offered by *Enterprise*.

2.2.1 Crash course in autonomy

During the month of January, 2015, the MERS group (including the authors), taught a hands on, intro level autonomy course aimed at MIT freshmen and sophomores, titled "Crash Course in Autonomy." The course was broken into five different modules covering different aspects autonomous systems. In each module, students were provided an intro lecture, an advanced lecture, and a lab assignment to apply the technologies covered in the module to simulated and real ARDrone quadrotors.

For the hands-on lab assignments, the students used different instantiations of *Enterprise* appropriate for the lab at hand. For each of the five modules, *Enterprise* was used in the following configurations:

- *Module 1: Scripting* - In this configuration, students were required to write an RMPL program that compiled directly into an executable TPN. *Enterprise* was configured to only use Pike as a dispatcher.

- *Module 2: Localization - Enterprise* was unchanged from Module 1. Instead, the control layer was augmented to include vision-based localization and mapping.

- *Module 3: Path planning - Enterprise* was configured to include basic visibility graph and grid-based path planners. Students wrote RMPL programs in terms of traversals between named locations instead of specifying the coordinates of a trajectory to follow.

- *Module 4: Activity planning - Enterprise* was configured to include the tBurton activity planner [11] in addition to the path planners. Students wrote RMPL programs in terms of goals instead specifying which actions to take.

- *Module 5: Scheduling - Enterprise* was configured to include the Kirk planne [12] in addition to the path planners. Students wrote RMPL programs in terms of which actions to perform, but were allowed to specify choices between different action sequences that achieved the same goal.

As this was an introductory course, risk-aware planners and schedulers were not covered.

Sixteen students participated in this class. The majority of the students self-reported as having little to no experience with autonomous systems, but also stated they found *Enterprise* easy to use and understand.

2.2.2 WHOI

The *Enterprise* architecture has also been deployed to control a Slocum glider autonomous underwater vehicle (AUV) owned and operated by scientists at the Woods Hole Oceanographic Institution (WHOI). This capability was demonstrated during a technology validation expedition on the R/V Falkor in the Scott Reef lagoon in the Timor Sea from March 24 to April 6, 2015. In this demonstration, *Enterprise* was used as a decision support system to plan for glider operations in the presence of five other AUVs and the Falkor itself. The human operators used *Enterprise* to plan a series of observations of target regions in between surfacing for data communication and plan underwater paths for the observations.

In the demonstration, *Enterprise* was used with RMPL input, the Kirk planner, and a simplified, risk-aware path planner. Due to technical constraints imposed by the glider control system, Pike was not used as a dispatcher. Instead, the executable TPN was translated directly into the glider's scripting language.

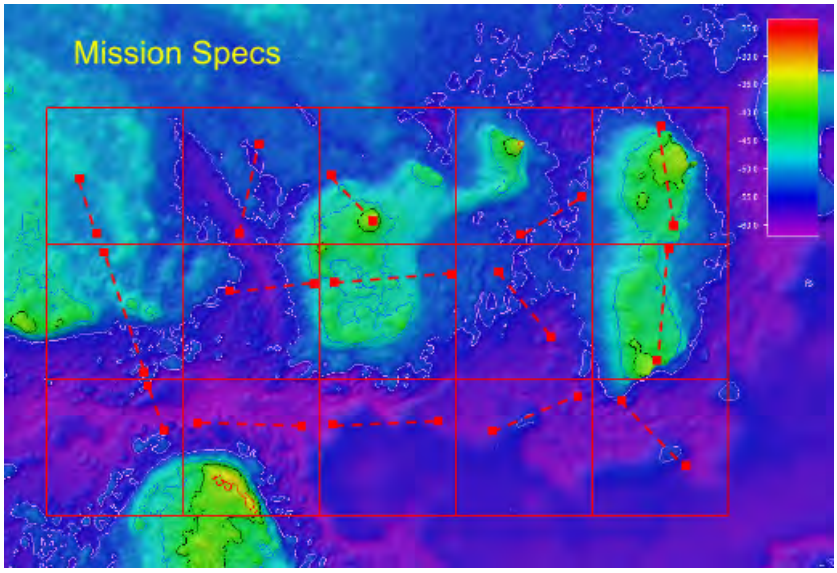


Figure 2.3: Mission goals for the Slocum glider during the Scott Reef deployment.

Figure 2.3 illustrates the mission goals for the glider deployed during the expedition. Operators discretized a specific area of the lagoon in 15 regions of interest, cells, to be visited by the glider. Each cell was assigned a priority and a path (red dashed line) for the glider to traverse. All AUVs on the deployment shared the cells, but each had unique goals in each cell. In order to avoid collisions, a constraint was placed on the AUVs that no more than one AUV could occupy a cell at a time.

At the beginning of the deployment, the path planner was used without Kirk to plan transits for nine days in initial testing. At the time of the cruise, the path planner used a simplified dynamics model and relied on the operator for risk allocation. Kirk and the path planner were then used in conjunction to successfully plan for two days of eight hour operations for the glider. The activity planner efficiently 1) selected subset of science goals with highest return based on science preference, and 2) ordered and scheduled visitation to respect the aforementioned constraints. Ocean currents in Scott Reef changed frequently and posed a challenge for the AUVs deployed during the expedition. The path planning component successfully planned safe routes around the reef. Moreover, we demonstrated the executives capability to support re-planning after each glider surface activity. To the best of our knowledge, a Slocum glider has never before been used inside a reef before, due to the challenges present in that environment.

Chapter 3

Programming risk-aware missions with cRMPL

This project developed a chance-constrained, reactive model-based programming language (cRMPL), that allows the desired behavior of autonomous systems to be specified at a high-level of abstraction. It extends the original RMPL [13] with the following features: I) sensing actions, in the form of probabilistic choice nodes; II) probabilistic temporal uncertainty; III) safety guarantees in the form of chance constraints; and IV) state constraints.

Mission specifications in cRMPL offer flexibility in the choice of action sequences used to reach goals, which are exploited during execution to achieve robustness. In cRMPL, like traditional programs, time-evolved behavior is specified using standard control constructs, including parallel and sequential execution, conditional execution, iteration, contingent, and timed execution. The latter enables the specification of time-critical missions.

Unlike traditional languages, cRMPL bounds the risk of execution failure, by allowing a chance constraint to be specified over any cRMPL (sub)expression. This constraint specifies a maximum probability that that expression will fail to terminate successfully. To improve robustness, cRMPL includes several constructs for introducing choices that the executive makes at run-time, in order to adapt to delays and failures. This includes decision-theoretic choices between functionally-equivalent procedures and bounds on procedure execution time. Programs in cRMPL can also be elevated from specifying action sequences to specifying desired state evolutions, by introducing state constraints as program primitives. The executive maps states to actions by continuously planning using a set of action models.

3.1 Features of cRMPL

Our implementation of cRMPL is in the form of an extension of Python, a widely used, general-purpose programming language. For that reason, cRMPL programs are represented as objects of the *RMPyL* class, which stands for "RMPL in Python". A list of selected cRMPL features follows:

- since cRMPL is a Python module, anything available in Python can be used to manipulate cRMPL objects, such as list comprehensions, dictionaries, recursion, file I/O, network interfaces, etc.;
- cRMPL only depends on the availability of a Python interpreter (either Python 2 or 3) and its standard libraries, therefore being cross-platform (Windows, Mac, and GNU/Linux);
- cRMPL is useful for sequencing (describing a temporal program programmatically), as well as for modeling tasks;
- cRMPL is built around the concept of *Episodes* (Section 3.2) and their compositions. Therefore, it is very easy to write libraries of cRMPL subroutines that can be composed to form more complex cRMPL programs;
- cRMPL programs can be translated into Probabilistic Temporal Plan Networks (pTPN's) [1] and dispatched by the pKirk executive, both developed in the context of this grant.

3.2 Episodes

The core building block of cRMPL is the *episode* (see Figure 3.1). An episode can be intuitively understood as a period of time during which an activity must be performed, while respecting a family of state, temporal, and chance constraints. If the activity can be directly executed by the autonomous system under control, we call the episode *primitive*. Alternatively, if the activity to be performed within an episode consists of a combination of other episodes, we call the outer episode *composite*. The ways in which episodes can be combined in cRMPL are explained in Section 3.4.

More formally, an episode E is a tuple $<s, e, A, \mathbb{S}, \mathbb{T}, \mathbb{C}>$, where s and e are, respectively, the temporal events marking the start and end of E; A is either a primitive activity (can be readily executed by the autonomous agent) or another

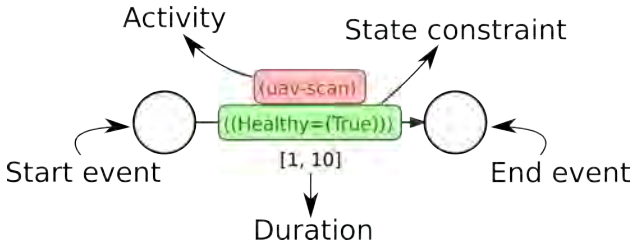


Figure 3.1: Example episode specifying that an unmanned aerial vehicle (UAV) should scan an area for a period between 1 and 10 time units, while making sure that it maintains itself in a healthy state. If *uav-scan* can be directly executed by the UAV, this would be a primitive episode. Otherwise, if *uav-scan* requires a combination of more fundamental operators (according to the operations in Section 3.4), then this episode would be composite.

episode; and $\mathbb{S}, \mathbb{T}, \mathbb{C}$ are, respectively, sets of state, temporal, and chance constraints that should hold during the period of time E is being executed. In cRMPL, episodes are instances of the *Episode* class.

The next section provides further details about the types of constraints that can be represented within episodes.

3.3 Constraints in cRMPL

Constraints in cRMPL, regardless of their specific type, are split into two groups: *model* and *user-defined*. This characterization is particularly important in the context of chance constraints (Section 3.3.3), since they can only be imposed over user-defined state and temporal constraints. A *model* constraint is one that stems from physical limitations of the system at hand, and, therefore, always holds. One can mention as examples of model constraints the conservation of flow in network problems; the degrees of freedom of a robotic arm; and the maximum speed that a vehicle can attain. On the other hand, *user-defined* constraints, as their name says, are externally imposed on the system by the cRMPL programmer to cause it to behave appropriately. For instance, speed limits on highways are user-defined constraints that dictate the desired behavior for drivers on that road. Similarly, a cRMPL programmer could impose the state constraint "stay away of no-fly zones" in the episode in Figure 3.1.

With this distinction in mind, the following sections provide further details

about the types of constraints that are supported within episodes in cRMPL.

3.3.1 Temporal constraints

Three basic types of temporal constraints are supported in cRMPL: simple temporal constraints (STCs) [14]; STCs with uncertainty (STCUs) [15]; and probabilistic STCs (pSTCs) [4, 5][1].

An STC over two temporal events e_1 and e_2 is a tuple $<e_1, e_2, l, u>$ imposing the constraint $e_2 - e_1 \in [l, u]$, $l \leq u$, $l, u \in \mathbb{R}$. For an STC, both e_1 and e_2 can be freely chosen.

An STCU is very similar to an STC: it is given by a tuple $<e_1, e_2, l, u>$, $l \leq u$, $l, u \in \mathbb{R}^+$. However, for an STCU, the value of e_2 is uncontrollable: it will be chosen by the environment during execution so that $e_2 - e_1 \in [l, u]$, but its specific value cannot be chosen beforehand.

A pSTC extends STCUs by allowing random variables with known probability distributions to describe the temporal distance between two events. More formally, a pSTC is a tuple $<e_1, e_2, v>$ such that $e_2 = e_1 + v$, where v is a random variable. As with STCUs, the specific value of e_2 is not directly controllable: it is chosen by the environment according to the value of e_1 and the probability distribution of v.

3.3.2 State constraints

Let X be a vector of state variables. The current implementation of cRMPL supports discrete state variables over finite value domains, and numerical state variables over continuous ranges of values. For those, two types of states constraints are available:

- general linear constraints of the form $AX_n \lesseqgtr b$, where A, b are constant matrices of appropriate dimensions and X_n is the subset of X composed of numerical state variables;

- and assignment constraints $X = c$, where c is a vector of constants.

[1] Disjunctive and conditional temporal constraints of different types can be represented as combinations of these different types of simple temporal constraints with the choice operators from Section 3.4.3.

Alternatively, one can also specify a Boolean constraint-checking function $f_S(X)$, which checks if the vector of state variables X satisfies the state constraints $S \subseteq \mathbb{S}$.

There is also a distinction in terms of the period of time during which state constraints should hold:

at start : state constraints that should hold by the time the start event of an episode is executed;

at end : same as *at start*, but for the end event of an episode;

during : state constraints that should hold during the whole period between the start and end events, but not necessarily at the extrema;

overall : state constraints that should hold at all previously mentioned periods.

3.3.3 Chance constraints

The ability to impose chance constraints on episodes is one of the most important features of cRMPL. In essence, a chance constraint provides a bound Δ on the probability of a set of *user-defined* constraints R, $R \subseteq (\mathbb{S} \cup \mathbb{T})$, being violated during the execution of the episode. Therefore, a chance constraint is a tuple $C = <R, \Delta>$. Model constraints are not included in chance constraints because they are trivially satisfied by the underlying physics.

3.4 Composing episodes in cRMPL

As in the original RMPL, cRMPL subroutines can be hierarchically combined into composite episodes using sequence, parallel, and choice operators. There is also iteration, which is built on top of sequence and choice. As previously mentioned, cRMPL programs are represented as instances of the *RMPyL* class, which are capable of specifying complex temporal behavior by combining episodes through the operations described in this section.

3.4.1 Sequence composition

In a *sequence* composition of episodes (Figure 3.2), one is executed immediately after the other, with an optional controllable slack between them represented by a

$[0, \infty]$ STC. In cRMPL, a list of episodes can be composed in sequence through the class method *RMPyL.sequence*. Alternatively, pairs of episodes can be composed in sequence by the overloaded binary operator "*".

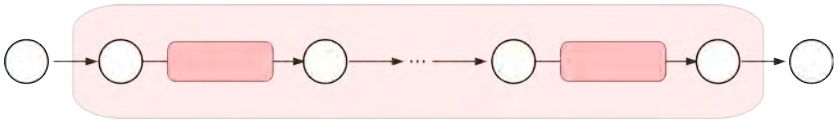


Figure 3.2: Excerpt of a pTPN depicting a sequential composition of episodes.

3.4.2 Parallel composition

In a *parallel* composition of episodes (Figure 3.3), there is a common event where the parallel composition starts, followed by the simultaneous execution of all episodes in the composition, and a common end event where all parallel execution branches come to an end. In cRMPL, a list of episodes can be composed in parallel through the class method *RMPyL.parallel*. Alternatively, pairs of episodes can be composed in sequence by the overloaded binary operator "+".

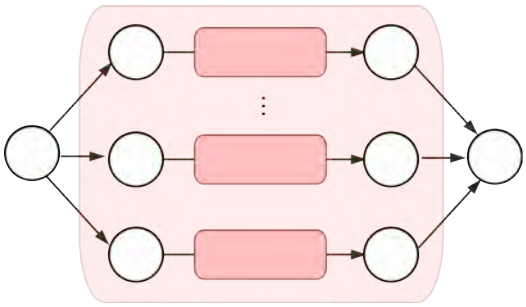


Figure 3.3: Excerpt of a pTPN depicting a parallel composition of episodes.

3.4.3 Choice composition

A *choice* composition is similar to a parallel one in terms of structure, but only one of the branches is ever executed. Therefore, it represents a disjunction. Choices are pictorially represented as double circles, as shown in Figure 3.4. Choices

are either controllable (assigned by the control program) or uncontrollable (assigned by an external agent, such as a sensor reading). Uncontrollable choices can either be non-deterministic and with no associated probability distribution; or probabilistic, where there is a probability associated with each possible outcome. Choice compositions are implemented in cRMPL by the class method *RMPyL.choose*. Since controllable choices are often used to represent decision by the program executive, while uncontrollable choices usually represent sensor readings and other types of environmental observations, cRMPL programs also have the syntactic-sugar constructs *RMPyL.decide* and *RMPyL.observe* to represent, respectively, controllable and uncontrollable choices.

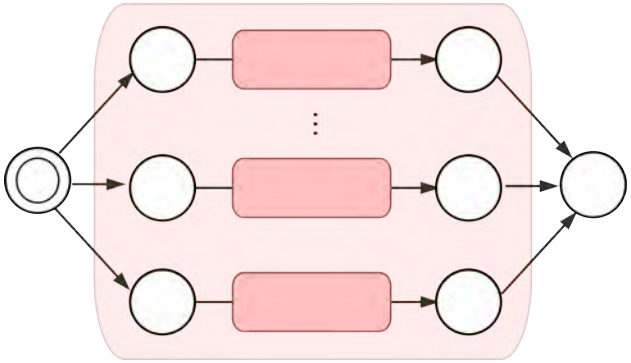


Figure 3.4: Excerpt of a pTPN depicting a choice (disjunction) composition of episodes.

3.4.4 Iteration composition

Iterations in cRMPL are implemented by the class method *RMPyL.loop*. It is recursively defined using the previously described sequence and choice operators, modeling the process of choosing to execute an activity one more time, and terminate the looping behavior. Iterations constructed with controllable choices give the program executive flexibility to execute an activity one or more times (up to a maximum number of repetitions), should that be beneficial to the mission. On the other hand, loops with uncontrollable choices can be used to represent an externally-controller looping behavior.

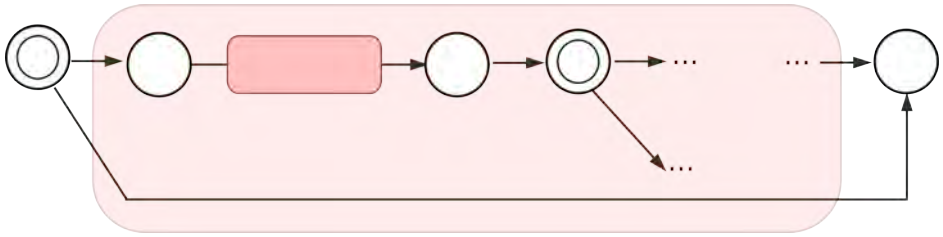


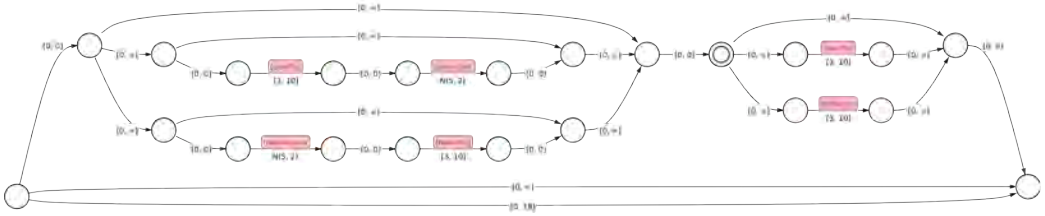
Figure 3.5: Excerpt of a pTPN depicting an iterative composition of episodes. At the beginning of each iteration, there is a choice between executing the loop once more, or ceasing the iteration.

3.5 Simple UAV scenario in cRMPL

Figure 3.6 presents a “Hello World” example showing how cRMPL can be used for modeling, as well as for programming desired temporal behavior.

```
from rmpyl.rmpyl import RMPyL,Episode,ChanceConstraint
class UAV: #UAV class that generates episodes
  def fly(self):
    return Episode(duration={'ctype':'controllable',
                             'lb':3,'ub':10},
                             action='(fly)')
  def scan(self):
    return Episode(duration={'ctype':'controllable',
                             'lb':1,'ub':10},
                             action='(scan)')
hello = UAV(); uav = UAV()
pr = RMPyL()
pr.plan = pr.sequence(
            pr.parallel(
              pr.sequence(
                   hello.scan(),
                   hello.fly()),
              pr.sequence(
                   uav.fly(),
                   uav.scan())),
            pr.decide({'name':'UAV-choice',
                       'domain':['Hello','UAV'],
                       'utility':[5,7]},
                       hello.fly(),
                       uav.fly()))
tc=pr.add_overall_temporal_constraint(ctype='controllable',
                                      lb=0.0,ub=18.0)
cc_time = ChanceConstraint(constraint_scope=[tc],risk=0.1)
pr.add_chance_constraint(cc_time)
```

(a) "Hello World" example in cRMPL.



(b) Corresponding pTPN.

Figure 3.6: "Hello World" example in cRMPL and its corresponding representation as a pTPN.

Chapter 4

Risk-bounded consistency of Probabilistic Temporal Plan Networks

Autonomous agents often are not adopted in highly uncertain environments due to the risk of mission failure and loss of vehicles. Prior work on contingent plan execution addresses this issue by placing bounds on uncertain variables and by providing consistency guarantees for a 'worst-case' analysis, which tends to be too conservative for real-world applications. This chapter presents work that unifies features from risk-sensitive trajectory optimization and high-level plan execution in order to extend existing guarantees of consistency for conditional plans to a chance-constrained setting. The result is a set of efficient algorithms for computing plan execution policies with explicit bounds on the risk of failure. To accomplish this, we introduce Probabilistic Temporal Plan Networks (pTPN's), which improve upon previous formulations by incorporating probabilistic uncertainty and chance-constraints into the plan representation. We develop a novel method to the chance-constrained strong consistency problem, by leveraging a conflict-directed approach that searches for an execution policy that maximizes reward while meeting the risk constraint. Experimental results indicate that our approach for computing strongly consistent policies has an average scalability gain of about one order of magnitude, when compared to current methods based on chronological search.

4.1 Representing uncertainty in contingent temporal plans

Real-world environments are inherently uncertain, causing agents to inevitably experience some level of risk of failure when trying to achieve their goals. Instead of neglecting the existence of risk or overlooking the fact that unexpected things might have significant impacts on a mission, it becomes key for autonomous systems trusted with critical missions to have a keen sensitivity to risk and to be able to incorporate uncertainty into their decision-making.

In this chapter, we address the problem of extracting execution policies with risk guarantees from contingent plans with uncertainty. The current practice for ensuring safety in these missions requires groups of engineers to reason over a very large number of potential decisions and execution scenarios that might unfold during execution, which is a challenging, time-consuming, and error prone process. Given a description of a contingent plan, several different approaches in the literature developed notions of consistency by representing uncertainty as set-bounded quantities, i.e., as intervals of values with no associated probability distribution [16–19]. Nevertheless, in other to guarantee feasibility in all possible scenarios, consistency-checking algorithms based on set-bounded uncertainty end up performing a worst-case analysis. When considering situations where uncertainty causes small plan deviations around otherwise "nominal" values, these set-bounded consistency criteria work well and output robust, albeit conservative, execution policies. However, they have difficulties handling problem instances where uncertainty can potentially lead the system to very hard or even irrecoverable scenarios, often returning that no robust execution policy exists. This is most certainly undesirable, since reasonable amounts of risk can usually be tolerated for the sake of not having the autonomous agent sit idly due to its absolute "fear" of the worst.

The work presented in this chapter improves upon the state-of-the-art on conditional plan execution by extending the notions of weak and strong plan consistency to a risk-bounded setting and providing efficient algorithms for determining (or refuting) them. These risk bounds are also known as chance-constraints [20]. Weak and strong consistency are useful concepts when planning missions for agents whose embedded hardware has very limited computation and telecommunication power, making it hard for them to come up with solutions 'on the fly' or for remote operators to intervene in a timely fashion. Chance-constrained weak consistency (CCWC) is a useful concept for missions where agents operate

in static or slow changing environments after an initial scouting mission aimed at reducing plan uncertainty. Chance-constrained strong consistency (CCSC), on the other hand, removes the need for a scouting mission and tries to determine the existence of a solution that, with probability greater than some threshold, will succeed irrespective of the outcomes of uncertainty in the plan. Strong consistency is clearly more conservative, but it is appealing to mission managers because strongly consistent policies require little to no onboard sensing and decision making, greatly reducing the agents' complexity and costs. They also reduce or completely eliminate the need to coordinate between multiple agents. Finally, the robustness of a strongly consistent policy makes it easier to check by human operators before it is approved for upload to the remote agent.

We introduce Probabilistic Temporal Plan Networks (pTPNs) as our representation of contingent plans. Our pTPN representation holds a lot of similarities with Temporal Plan Networks with Uncertainty (TPNU) [2, 17], Conditional Temporal Plans (CTPs) [16], Disjunctive Temporal Problems with Uncertainty (DTPU) [18], and the Conditional Simple Temporal Network with Uncertainty (CSTNU) [19], but extends them in two important ways. First, pTPNs allow uncontrollable choices (discrete, finite domain random variables) to have their joint distributions described by a probability model, as opposed to a purely set-bounded uncertainty representation in DTPUs and CTPs. Second, pTPNs allow the user to specify admissible risk thresholds that upper bound the probability of violating sets of constraints in the plan, a missing feature in CTPs, DTPUs, and TPNUs. The latter extension is an important improvement of pTPNs over previous representations when modeling many real-world problems, where risk-free plans that are robust to all possible uncontrollable scenarios are often unachievable. Our pTPNs are compiled from contingent plans descriptions in cRMPL (Chapter 3).

4.2 A conflict-directed approach to risk-bounded plan consistency

Our algorithms reason quantitatively about the probability of different random scenarios and explore the space of feasible solutions efficiently while bounding the risk of failure of an execution policy below a user-given admissible threshold. While state-of-the-art methods in the conditional and stochastic CSP literature rely on a combination of chronological (depth-first) search and inference in the space of contingencies in order to quickly find satisficing solutions [21–25],

in this work we introduce a "diagnostic" approach based on Conflict-Directed A^* (CDA*) [26]. By continuously learning subsets of conflicting constraints and generalizing them to a potentially much larger set of pathological scenarios, our algorithms can effectively explore the space of robust policies in best-first order of risk while ensuring that it is within the user-specified bound. For the problem of extracting a strongly consistent policy from a contingent plan description, our numerical results showed significant gains in scalability for our approach.

Here we motivate the usefulness of chance-constrained consistency on a very simple commute problem, whose pTPN representation is given in Figure 4.1. We start at home and our goal is to be at work for a meeting in at most 30 minutes. Circles represent the start and end events of temporally-constrained activities called episodes. Simple temporal constraints [14] are represented by arcs connecting temporal events and represent linear bounds on their temporal distance. For simplicity, we assume that we are given only three possible choices in this pTPN: we can either ride a bike to work, drive our car, or stay home and call our employer saying that we will not be able to make it to work today. The rewards (R values) associated with each one of these choices in Figure 4.1 correspond to how much money we would make that day minus the cost of transportation. Uncontrollable choices are depicted in Figure 4.1 by double circles with dashed lines. These are random, discrete events that affect our plan and whose probability model is also given in Figure 4.1.

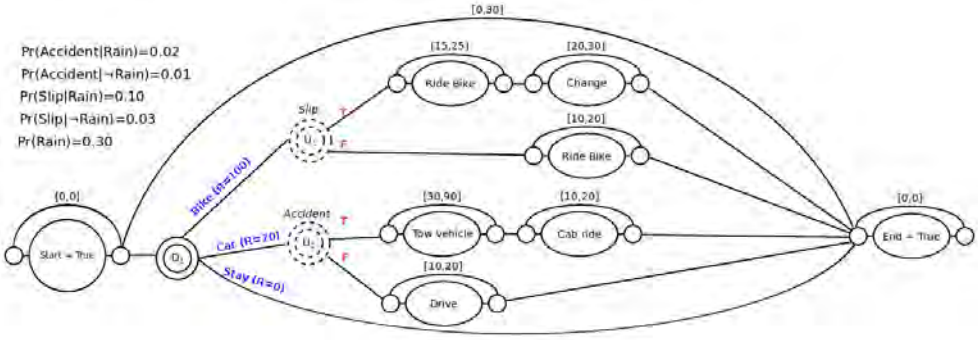


Figure 4.1: A pTPN for a simple plan to get to work

In this example, the uncontrollable choices model what might "go wrong" during plan execution and the impact of these unexpected events on the overall duration of the plan. For example, if we decide to ride a bike to work (the option with the highest reward), there is the possibility that we might slip and fall. This

event has a minor effect on the duration of the ride, but would force us to change clothes at our workplace because we cannot spend the day in a dirty suit. Since we only have 30 minutes before the meeting starts, the uncontrollable event of slipping would cause the overall plan to be infeasible. A similar situation happens if we choose to drive our car and happen to be involved in an accident.

By ignoring probabilities and using a consistency checker for the pTPN in Figure 4.1 based on a set-bounded representation of uncertainty, we would realize that the pTPN is guaranteed to be consistent. Unfortunately, unless we had a way of telling ahead of time whether we would slip from the bike or be in a car accident, the suggested policy would be to always stay home! This is because, for the choice of riding a bike or driving our car to work, there are uncontrollable scenarios that cause the plan to fail, causing the set-bounded consistency checker to fall back to the safe, albeit undesirable, choice of staying at home. This clearly disagrees with our common sense, since people try to achieve their goals while acknowledging that uncontrollable events might cause them to fail. Next, we show how our chance-constrained approach would produce execution policies that agree with what we would expect a "reasonable" agent to do.

Let's consider the case where we accept that our plan might fail, as long as the risk Δ is no more than 2%. Given that riding a bike is the option with the highest reward, our algorithm would deem bike ridding the most promising and would start by checking if choosing to ride a bike meets the chance-constraint $\Delta \leq 2\%$. If there existed a feasible activity schedule satisfying the temporal constraints for both values of $Slip$, we could pick this schedule and our risk of failure would be zero, which is clearly less than our risk bound. However, our algorithm concludes that the scenario $Slip = True$ is inconsistent with the overall temporal constraint of arriving at the meeting in less than 30 minutes, so there must exist a nonzero risk of failure in this case. According to the model in Figure 4.1, the probability of having slip is $\Pr(Slip) = 5.1\%$, so riding a bike does not meet the chance-constraint $\Delta \leq 2\%$. The next best option is riding a car, where now we are subject to the uncontrollable event of being in a car accident. Following a similar analysis, we conclude that the risk of our plan being infeasible in this case is $\Pr(Accident) = 1.3\%$, which meets the chance-constraint. Therefore, our algorithm would advise us to drive to work within the temporal bounds shown in Figure 4.1 for the case where no accident happens. We claim that this chance-constrained type of reasoning approximates the decision making process of mission managers much better than its set-bounded alternative, since operators need to commit to plans with acceptable levels of risk in order to extract some useful output from the remote explorer.

It is worth noticing that choosing $\Delta < 1.3$ would have made staying at home the only feasible alternative. Hence, as in the set bounded approach, a chance constraint may still be too conservative to allow for a feasible solution. Moreover, if the overall temporal constraint of 30 minutes in Figure 4.1 were relaxed to 35 minutes, our algorithm would have been capable of finding a risk-free scheduling policy for its first choice of riding a bike. This is because, in this case, there would exist a feasible schedule satisfying all temporal constraints on the upper side of the pTPN, i.e., temporal constraints activated by both $Slip = True$ and $Slip = False$.

This example highlights a few key elements of our approach. First, we divide the problem into generating a candidate "plan" as an assignment to the controllable choices and testing the plan against the chance constraints. Second, candidates are enumerated in best-first order based on reward. Third, testing feasibility of a chance-constraint is a fundamental task, in which estimating risk is costly. We frame risk estimation as a process of enumerating the most likely sets of scenarios that incur and do not incur risk (called conflicts and kernels, respectively). We observe that this can be formulated as a symptom-directed, "diagnostic" process, allowing us to leverage an efficient, conflict-directed best-first enumeration algorithm, Conflict-Directed A^* (CDA*) [26], to generate kernels and conflicts, and hence feasible and infeasible scenarios. A detailed description of this work can be found at [1].

Chapter 5

Model-based generation of risk-aware plans

Similar to conventional programming languages, using cRMPL (Chapter 3) to specify safe autonomous behavior at a high level of abstraction is done *declaratively*, i.e., the programmer is responsible for the explicit enumeration of all sequence and parallel composition of episodes, along with all decisions (controllable choices) and observations (uncontrollable choices) in the program. Once it is available, this program can be checked for risk-bounded temporal consistency using the methods described in Chapter 4. There are, however, two important caveats related to this approach. First, the computational cost of checking consistency of cRMPL programs and the size of their corresponding pTPN's grow exponentially with the number of choices. Second, in application domains where there is significant flexibility in terms of the decisions an autonomous agent can make, and the number of possible observations that it might receive from the environment, thinking through the sequential *act-observe-act-*$\cdots$ process declaratively can be too challenging of a task for a human programmer.

This chapter introduces Risk-bounded AO* (RAO*), a heuristic forward search algorithm capable of automatically generating cRMPL programs from chance-constrained POMDP (CC-POMDP) models, a novel variant of Partially Observable Markov Decision Processes (POMDP's) that we propose to allow autonomous agents operating under uncertainty to optimize expected performance while bounding the risk of violating safety constraints. In the development of RAO* (detailed technical description available in Appendix A), we perform a systematic derivation of execution risk in POMDP domains, improving upon how risk was previously handled in the constrained POMDP (C-POMDP) literature. In addition

to the utility heuristic used to guide RAO* towards cRMPL programs with better performance, our algorithm leverages an admissible execution risk heuristic to quickly detect and prune overly-risky cRMPL program branches, therefore enabling their early pruning. In Chapter 7, we show how cRMPL programs generated with RAO* can be hierarchically combined within user-specified cRMPL programs in a hybrid declarative/generative approach for describing safe autonomous behavior.

5.1 The need for handling risk in planning domains with uncertainty

Partially Observable Markov Decision Processes (POMDPs) [27] have become one of the most popular frameworks for optimal planning under actuator and sensor uncertainty, where POMDP solvers find policies that maximize some measure of expected utility [28, 29].

In many application domains, however, performance is not enough. Critical missions in real-world scenarios require agents to develop a keen sensitivity to risk, which needs to be traded-off against utility. For instance, a search and rescue UAV should maximize the value of the information gathered, subject to safety constraints such as avoiding dangerous areas and keeping sufficient battery levels. In these domains, autonomous agents should seek to optimize expected reward while remaining safe by deliberately keeping the probability of violating one or more constraints within acceptable levels. Unsurprisingly, attempting to model risk bounds as negative rewards leads to models that are over-sensitive to the particular penalty value chosen, and to policies that are overly risk-averse or overly risk-taking [30]. Therefore, to accommodate the aforementioned types of scenarios, new models and algorithms for *constrained* MDPs have started to emerge, which handle chance constraints explicitly.

Research has mostly focused on fully observable constrained MDPs, for which non-trivial theoretical properties are known [31, 32]. Existing algorithms cover an interesting spectrum of chance constraints over secondary objectives or even execution paths, e.g., [33–35]. For constrained POMDPs (C-POMDP's), the state of the art is less mature. It includes a few suboptimal or approximate methods based on extensions of dynamic programming [36], point-based value iteration [37], approximate linear programming [38], or on-line search [30]. Moreover, the modeling of chance constraints through unit costs in the C-POMDP literature has a num-

ber of shortcomings, such as requiring constraint violations to be fully observable and cause program execution to halt immediately, leading to conservatism.

5.2 Searching for optimal, risk-bounded cRMPL programs

Because cRMPL supports programs that control agents with hidden state (only partially observable through sensing actions), RAO* performs its search for a performance-maximizing, risk-bounded cRMPL program in the space of discrete *belief states*. Simply put, a discrete belief state consists of a list of possible hidden states that an autonomous agent (and its environment) might be at a given point in time, along with its associated probability (see top right corner of Figure 5.1).

Starting from an initial belief state b_0, RAO* explores the space of belief states using heuristic forward search by keeping track of two graphs: I) an *explicit search graph* G, whose nodes represent all belief states explored so far; and II) the *greedy graph* g, which is the subset of the explicit graph that currently contains our best estimate of the best-performing cRMPL program. Figure 5.1 shows the tree representation of an "act-and-observe" step in a cRMPL program, and Figure 5.2 shows how the information in Figure 5.1 can be represented in the form of a hypergraph. As shown in Figure 5.2, controllable choices in cRMPL are represented as actions in the hypergraph. Once the agent performs that action, the agent receives one of a family of possible sensor inputs (including "no input"). These correspond to uncontrollable choices in cRMPL, and are represented as hyperedges (associated to that particular action) in the hypergraph where RAO* performs its search. Figure 5.2 shows a portion of the explicit search graph G with a leaf node on the k-th layer being expanded.

Similar to AO* [39], RAO* searches for performance-maximizing cRMPL programs by propagating optimistic (upper bound) estimates of utility recursively from child to parent nodes in its explicit graph G. In addition to utility, RAO* introduces a novel heuristic propagation of the execution risk associated with a cRMPL program, therefore allowing it to compute increasingly precise estimates of how likely it is for program execution to deviate from safety and violate one or more user-specified constraints. As with utility, recursive propagation of optimistic (lower bounds, in this case) estimates of a cRMPL program's execution risk are performed from child to parent nodes in the explicit graph G - see equation (12) in Appendix A. These optimistic risk estimates are used to quickly detect

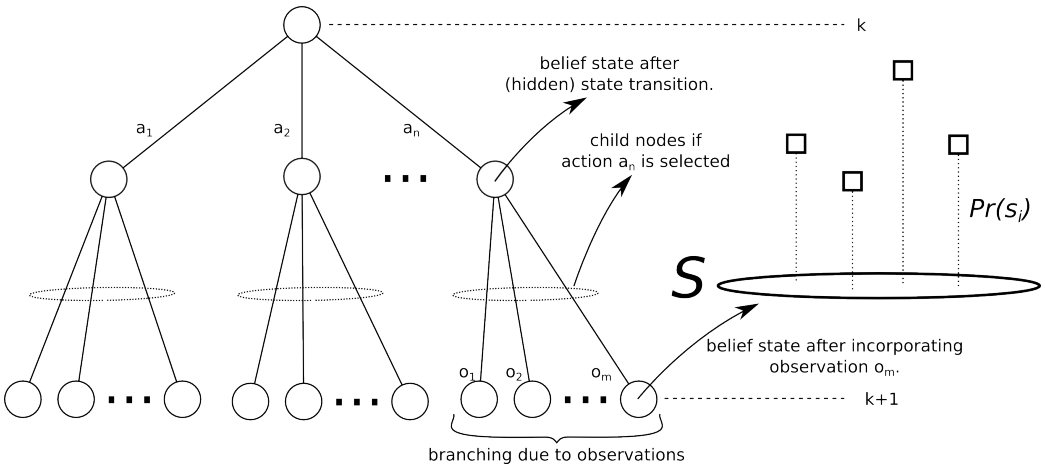


Figure 5.1: Detailed representation of the process of choosing among n different controllable choices in an cRMPL program ($a_1, a_2, \ldots, a_n$), and subsequently receiving one of m possible uncontrollable choices ($o_1, o_2, \ldots, o_m$). Nodes (circles) represent belief states, which are in turn represented as lists of possible hidden states and associated probabilities. Belief states after performing an action are called *prior* beliefs, while belief states that incorporate new observations are called *posterior* beliefs.

overly risk-taking branches in a cRMPL program during the search process, allowing for their early pruning and potential significant reduction of the search space. At any given point during the search, the greedy graph g of RAO* represents the current estimate of a contingent cRMPL that maximizes agent performance while strictly abiding to the risk-bounds defined by chance constraints.

Appendix A presents a detailed derivation of the RAO* algorithm, along with its pseudo-code and thorough numerical evaluation.

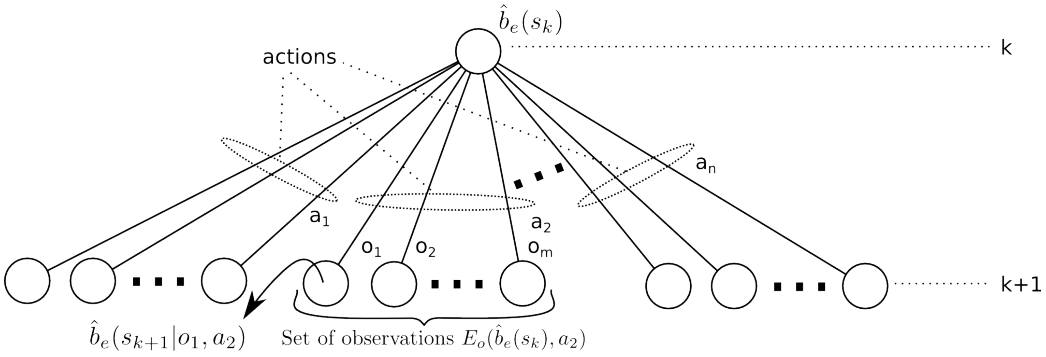


Figure 5.2: Representation of Figure 5.1 as a hypergraph node with several hyperedges associated to different actions. In a cRMPL program, actions correspond to controllable choices (decisions), while hyperedges are associated with the possible observations (uncontrollable choices) that the agent might receive from the environment upon executing an action.

Chapter 6

Chance-constrained optimal scheduling

In field deployment, there is often uncertainty about the exact timing of events caused by actions whose durations which are not controllable by the operator and not known a priori. For example, while the nominal flight time of a vehicle may be known, the actual flight time varies based on the conditions on the day. A reliable, robust scheduling scheme is thus required to take into account the uncertainty in durations in the mission.

The algorithm known as *Picard* deals with the risk-aware scheduling aspect of the overall problem. In contrast to traditional approaches, we use probabilistic representations of uncertainty, representing the values of uncontrollable durations with random variables. The distributions allow us to reason about the most likely ranges for the durations, and provide schedules which will meet the timing requirements with probabilistic guarantees without undue conservatism.

6.1 Problem Statement

In field deployment on critical missions, the cost of failing to meet timing constraints is often difficult to quantify. We must instead provide probabilistic guarantees for timeliness, accounting for the uncertain durations. In addition to robustness against constraint violation, the desirability of schedules may also depend on the time assignments: the quality of shallow water data may depend on the collection time due to the tidal cycle, and car sharing networks may require the inactive times of the cars to be low to maximize use of assets.

Descriptions and corresponding solution methods for such problems must thus have the following characteristics. First, the description must allow the specification of an utility function to be optimized. Second, the description must recognize stochasticity in durations with a probabilistic representation. Further, the problem description must allow rich expressions of constraints. For example, we must be able to describe requirements between the timing of two uncertain events when we schedule a traversal with uncertain duration to observe natural phenomena with uncertain timing. The scheduler must thus maximize utility while providing probabilistic guarantees of compliance with requirements.

We thus make use of the formalism described in [7] to define the underlying network describing the temporal constraints.

Definition 1. *(Probabilistic STN) Let:*

- **activated** *time-points* $b_i \in \mathbb{R}$ *be those assigned by the agent;*
- **received** *time-points* $e_i \in \mathbb{R}$ *be those assigned by the external world;*
- **free constraints** c_{xy} *(***Free***) be constraints of type* $(y - x) \in [l_{xy}, u_{xy}]$*, where* x*,* y *are time points; and*
- **uncertain duration** *(***uDn***)* $d_{xy} : \Omega \to \mathbb{R}$ *be random variables describing the difference* $(y - x) = d_{xy}(\omega)$*, where* y *is a received time point and* x *is an activated time point, for* $(\Omega, \mathcal{F}, P)$ *a probability space with sample space* Ω*,* σ*-algebra* $\mathcal{F}$ *and measure* P*.*

Then, $\mathcal{N}^+ = \langle X_b, X_e, R_c, R_d \rangle$ *defines a pSTN, with*

- $X_b = \{b_1, ..., b_B\}$ *the set of* $B \in \mathbb{N}$ *activated time-points;*
- $X_e = \{e_1, ..., e_E\}$ *the set of* $E \in \mathbb{N}$ *received time-points;*
- $R_c = \{c_{i_1 j_1}, ..., c_{i_C j_C}\}$ *the set of* $C \in \mathbb{N}$ *Frees; and*
- $R_d = \{d_{i_1 j_1}, ..., d_{i_G j_G}\}$ *the set of* $G \in \mathbb{N}$ *uDns;*

The pSTN allows us to express the controllable and uncontrollable events in a mission, as well as the requirements between them and the durations which are responsible for the uncertainty in the timing of events. We must find a schedule which optimizes an objective function, but also has a limited probability of failing to meet the constraints encoded in the pSTN. More formally, we must solve a chance-constrained probabilistic simple temporal problem (cc-pSTP), as defined below.

Definition 2. *(Chance-constrained pSTP)*
Given:

- $\mathcal{N}^+ = \langle X_b, X_e, R_c, R_d \rangle$*, a pSTN;*
- $\Delta_t \in [0, 1]$*, an upper bound on the risk of failure, for the set* R_c *of Frees; and*
- $V : \mathbb{R}^B \to \mathbb{R}$*, an objective function dependent on assignments to* X_b*;*

Find:

- $S^*_B \in \mathbb{R}^B$*, a schedule of* X_b *minimizing* V*;*

Subject to:

- $r_{R_c}(S^*_B) \leq \Delta_t$*, the probability of inconsistency bounded by* Δ_t*;*

In solving a cc-pSTP, we are required to find a timetable to each of the events whose timings we can control. The timetable must be optimal with respect to a predefined objective function, while meeting the temporal constraints set out in the pSTN with a probability greater than $1 - \Delta_t$.

6.2 Intuition for solution method

In this section we provide an intuition for the solution method. For the full solution method and the theoretical proofs and guarantees, the interested reader is directed to [7].

In general, it is not possible to guarantee against all eventualities. For example, it may be possible that a vehicle will take an infinite amount of time to arrive at a location, having broken down along the way. The key idea is to find a high probability subset of scenarios, and provide a schedule which will meet all timing constraints for the subset of scenarios.

We limit the scenarios we consider through risk allocation. We are given an upper bound on the probability of failure to meet the constraints. We can consider this the total risk allowed. We can spend this risk on the tail ends of the uncertain durations to limit the range of outcomes. By allocating the risk to the lower tail of an uncertain duration, we may find the outcomes for which the cumulative density function matches the allocated risk. Then, we can disregard any shorter outcomes

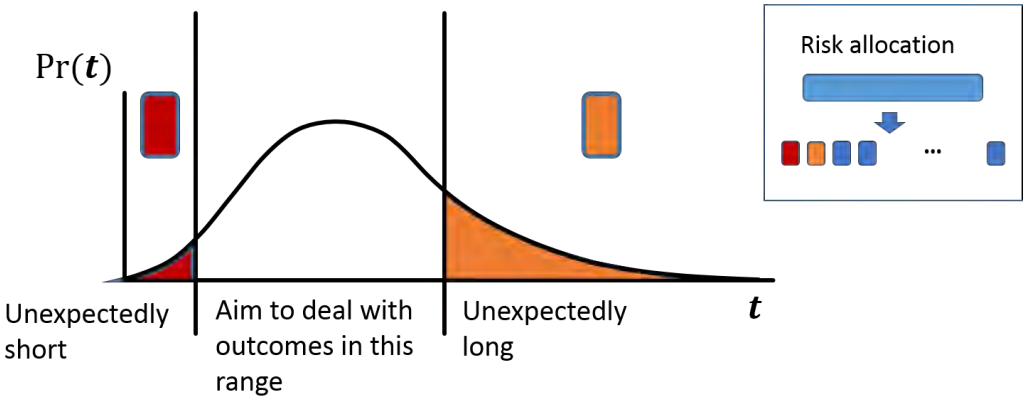


Figure 6.1: Risk allocation to find reasonable ranges of outcomes for uncertain durations.

for the uncertain duration - they can be considered expectedly short. A similar procedure is done with the upper bound.

In this way, we can define the unexpectedly long and short outcomes, and disregard them in our calculations. We are allowed to do this because the combined probability of any uncertain duration being in the unexpected regions is less than that allowed by the operator. We have thus derived set-bounded uncertainty for our temporal uncertainty. We are then able to call upon existing literature dealing with set-bounded temporal uncertainty, for example in [40]. This allows us to encode the cc-pSTP as a nonlinear optimization problem.

Chapter 7

Experimental validation

This chapter presents experimental validation, both in simulation and on real hardware, of the integrated components of *Enterprise* explained in this report. Detailed numerical analysis of the separate pieces can be found at the referenced publications, or at the appendices accompanying this report. Therefore, in this chapter, we focus on demonstrating the integrated deployments of our system, and evaluate the general applicability of our risk-aware architecture by demonstrating our algorithms in a number of different domains, ranging from planetary rovers and autonomous aircraft to robotic manufacturing.

The experiments in this section are organized as follows. In Section 7.1, we start by showing how cRMPL can be used for modeling and control purposes in a planetary exploration application involving the coordination of two rovers and a satellite. Next, Section 7.2 shows how cRMPL can be used to easily describe contingent plans in a robotic manufacturing setting, where a robot is actively trying to adapt to its human co-worker in order to achieve their common task goals. Finally, Section 7.3 shows how optimal cRMPL programs with sensing actions can be generated by RAO* and used to implement optimal, safe behavior of an autonomous aircraft tasked with finding hidden targets of interest.

7.1 Vehicle coordination under temporal uncertainty

Figure 7.1 depicts the planetary exploration scenario involved in this demonstration. Two autonomous rovers, *Spirit* and *Opportunity*, are tasked with the exploration of a number of predetermined locations on a region of Mars filled with obstacles. Once a rover arrives at a site, it must perform a number of science-

gathering activities whose temporal durations are uncertain, but bounded by upper and lower bounds hard-coded in the rover's science module. There is also uncertainty related to the traversal times between locations, and these are represented as random variables whose distributions are predicted based on the distance between the sites and the features of the terrain (Section 7.1.1 explains how these uncertainty models can be learned from data). Finally, both rovers have to go back to a relay site in order to transmit their findings to an orbiting satellite, which will be reachable by the rovers' antennas within a known time window. In order to maximize throughput, there is the added constraint that the rovers should communicate approximately at the same time with the satellite, but their transmissions should not overlap. Finally, since one would like to avoid any need of coordination between the two rovers, we require that there exists a precomputed schedule that satisfies all temporal constraints and is robust to the uncertainty in the temporal durations coming from the rover model.

Figure 7.2 shows the portion of cRMPL code describing the model for the different actions available to the rovers, along with their temporal durations and hierarchical composition. The *PySuluRMPyL* object implements pSulu [41, 42], a chance-constrained path planner developed in our group and called from within cRMPL to return trajectories that are safe when avoiding obstacles. These trajectories are converted into traversal episodes using cRMPL's composition constructs.

The cRMPL code in Figure 7.3 describes the temporal coordination of two science rovers. As we can see from the code, *Spirit* is responsible for gathering information about *minerals*, followed by a visit to *funny rock* and return to *relay*. In parallel, *Opportunity* must travel to the distant *alien lair*, perform its science, and then head to the relay location. The *tc_relay* constraint requires *Opportunity* to start sending its data no more than 10 seconds after *Spirit* has finished transmitting, and there is an overall temporal constraint $[1800, 2000]$ representing the 200 second window during which the satellite will be visible. A risk bound of 1% is placed on the violation of either of these two constraints, given the uncertainty associated with the temporal durations in this plan. In Figure 7.4, we see a pTPN representation of the cRMPL program in Figure 7.3.

The traversal episodes from to region shown in Figure 7.4 are each broken by pSulu into 10 intermediate segments, each one of them featuring a stochastic duration represented by a Gaussian random variable. We used Picard (Chapter 6) to generate a risk-bounded, strongly controllable (precomputed) schedule in 11.7 seconds, with the additional optimization that the schedule commands the rovers to initiate their activities as early as possible. From this example, one should

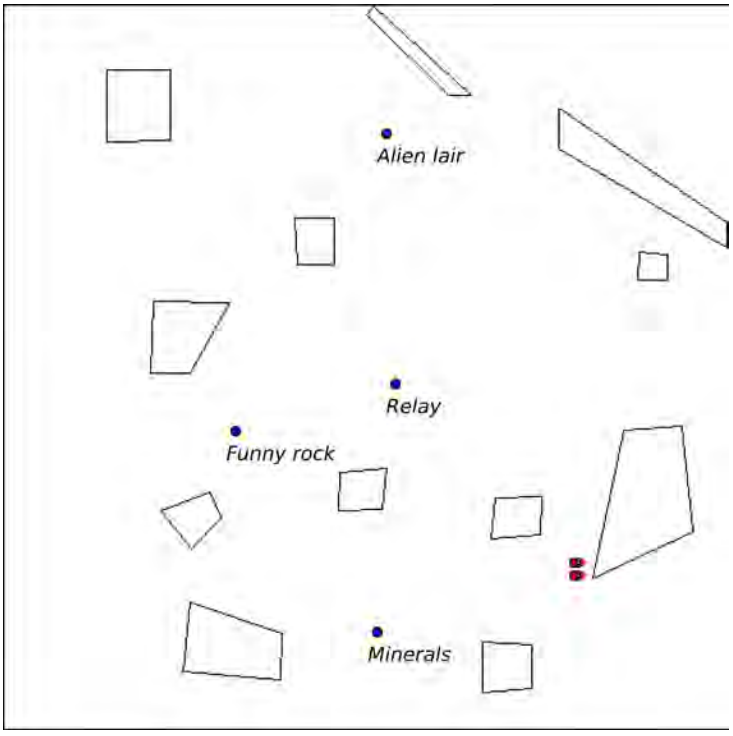


Figure 7.1: The *Mars rover* scenario featuring two robotic scouts (*Spirit* and *Opportunity*) that must explore different regions of the map and coordinate their communication with an orbiting satellite at a relay station. It is implemented in the MobileSim simulator.

notice how easy it is to model realistic temporal coordination missions in cRMPL with environmental uncertainty and risk bounds, and how efficiently Picard is able to leverage acceptable risk levels to compute schedules that are easy to verify, are robust to uncertainty, and offer hard guarantees that all mission requirements will be met with high probability (at least 99%, in this example).

7.1.1 Learning uncertain temporal duration models from data

The previously presented demonstration assumes knowledge about the temporal uncertainty associated with traversals, such as the one shown in Figure 7.5. There is the question, nevertheless, of how one can learn these traversal distributions from data, so that they can be incorporated into cRMPL models.

```
class Rover(object):
  """
  Simple RMPyL model for a Mars rover.
  """
  def __init__(self,name):
    self.name=name
    self.path_planner = PySuluRMPyL()

  def go_to(self,start,goal,risk,waypoints=10,time_horizon=200.0):
    """
    Returns the episode corresponding to the vehicle traveling.
    """
    self.rover_param['chance_constraint']=risk
    self.rover_param['waypoints']=waypoints
    self.rover_param['time_horizon']=time_horizon

    goto_ep = self.path_planner.plan_episode(start_state=start+(0.0,0.0),
                                             goal_state=goal+(0.0,0.0),
                                             parameters=self.rover_param,
                                             duration_type='gaussian',
                                             agent=self.name)
    return goto_ep

  def perform_science(self):
    """
    Returns the episode corresponding to the vehicle performing science
    experiments.
    """
    ps_ep=sequence_composition(
            Episode(duration={'ctype':'uncontrollable_bounded','lb':9,'ub':11},
                    action='(drill %s)'%(self.name)),
            Episode(duration={'ctype':'uncontrollable_bounded','lb':10,'ub':15},
                    action='(collect %s)'%(self.name)),
            Episode(duration={'ctype':'controllable','lb':5,'ub':30},
                    action='(process %s)'%(self.name)))
    return ps_ep

  def relay(self):
    """
    Returns the episode representing the rover sending data back to a satellite.
    """
    rel_ep= Episode(duration={'ctype':'controllable','lb':5,'ub':30},
                    action='(relay %s)'%(self.name))
    return rel_ep
```

Figure 7.2: cRMPL class modeling the actions of a science retrieval rover.

```
loc={'start':(8.751,-8.625),'minerals':(0.0,-10.0),
     'funny_rock':(-5.0,-2.0),'relay':(0.0,0.0),'alien_lair':(0.0,10.0)}

rov1 = Rover(name='spirit')
rov2 = Rover(name='opportunity')

prog = RMPyL(name='run()')
prog *= prog.parallel(
          prog.sequence(
            rov1.go_to(start=loc['start'],goal=loc['minerals'],risk=0.01),
            rov1.perform_science(),
            rov1.go_to(start=loc['minerals'],goal=loc['funny_rock'],risk=0.01),
            rov1.perform_science(),
            rov1.go_to(start=loc['funny_rock'],goal=loc['relay'],risk=0.01),
            rov1.relay(ep_id=rov1.name+'_relay')),
          prog.sequence(
            rov2.go_to(start=loc['start'],goal=loc['alien_lair'],risk=0.01),
            rov2.perform_science(),
            rov2.go_to(start=loc['alien_lair'],goal=loc['relay'],risk=0.01),
            rov2.relay(ep_id=rov2.name+'_relay')))

r1_rel = prog.episode_by_id(rov1.name+'_relay')
r2_rel = prog.episode_by_id(rov2.name+'_relay')

tc_relay = TemporalConstraint(start=r1_rel.end,end=r2_rel.start,
                              ctype='controllable',lb=0.0,ub=10.0)
prog.add_temporal_constraint(tc_relay)

tc=prog.add_overall_temporal_constraint(ctype='controllable',lb=1800.0,ub=2000.0)
cc_time = ChanceConstraint(constraint_scope=[tc,tc_relay],risk=0.1)
prog.add_chance_constraint(cc_time)
```

Figure 7.3: Program in cRMPL describing the temporal coordination between two science-retrieval agents.

Figure 7.4: pTPN representation of the control script in Figure 7.3.

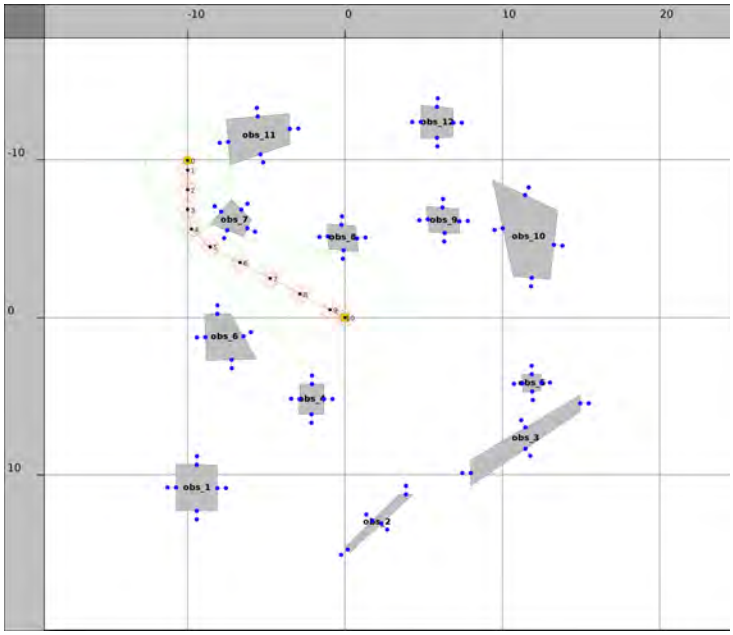


Figure 7.5: Example of a traversal generated by pSulu, a chance-constrained path planner.

In this demonstration, we made the assumption that traversal times between waypoints, as the ones shown in Figure 7.5, can be accurately predicted as a simple function of the Euclidean distance between them (length of the straight line connecting the two regions in space). To be more precise, we assumed the linear model $d(l) = al+b$, where l is the length of the line connecting the two waypoints; d is the duration of the traversal; and a and b are unknown parameters. In order to estimate a and b, we simulated hundreds of different traversals in the environment shown in Figure 7.1 and measured the time it took our path-following controller to drive a rover between those locations. For the i-th traversal performed out of a total of N, we recorded the pair (l_i, d_i). Then, we chose $\hat{a}, \hat{b}$, our best estimates of the parameters a and b, according to

$$\hat{a}, \hat{b} = \arg\min_{a,b} \sum_{i=1}^{N} (al_i + b - d_i)^2, \tag{7.1}$$

i.e., we chose $\hat{a}$ and $\hat{b}$ so as to minimize the variance of the predictor $\hat{d}(l) = \hat{a}l + \hat{b}$.

In order to estimate the variance of the prediction $\hat{d}(l)$ for an arbitrary length l, we compute the empirical variance of the samples (l_i, d_i) in a neighborhood around l. Figure 7.6 shows one such model learned with (7.1) from simulated data, where the upper and lower 3σ bounds for the variance are computed based on the worst-case empirical variance encountered on the dataset. Note that the exact same procedure could be used if real traversal data were available, therefore allowing (7.1) to be used with both real and synthetic data combined.

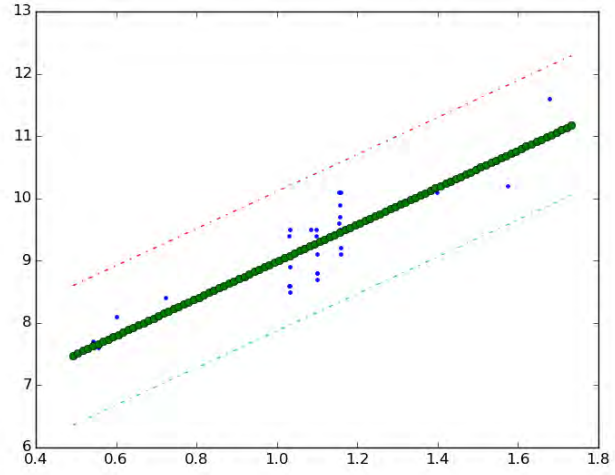


Figure 7.6: Stochastic traversal time model learned from data. The horizontal axis represents the Euclidean distance between waypoints, while the vertical axis represents the traversal time. The upper and lower 3σ bounds are computed based on the worst-case empirical variance encountered on the dataset.

7.2 Baxter cooperative manufacturing

In this integrated demo, we show how a non-expert user can interact with the Baxter robot in a collaborative task where the robot is constantly adapting to its human coworker, therefore requiring plans with embedded contingencies. In the proof of concept, the red and green blocks must be moved to their appropriate locations, as shown in Figure 7.7. At every given point in time, the robot observes the current state of the manufacturing task and acts accordingly. The videos at `http://`

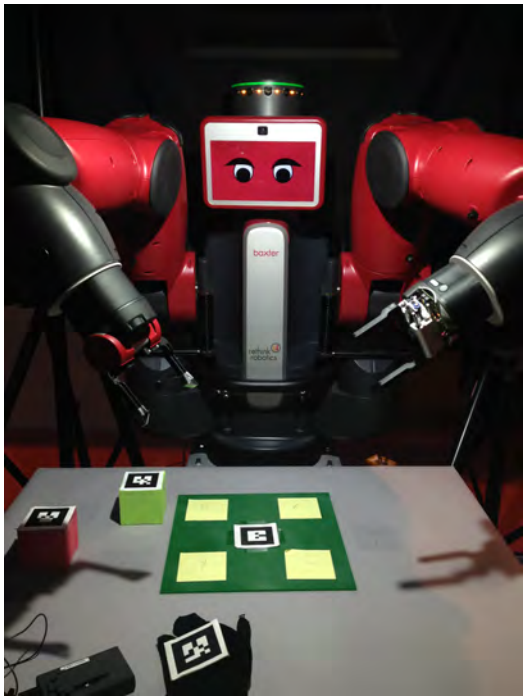


Figure 7.7: The Baxter workspace setup.

`people.csail.mit.edu/psantana/public/videos/AFOSR/` show two situations:

- in the nominal run, the human does not interfere with the robot, which proceeds to pick and place the blocks in sequence according to the execution policy;
- in the second run, the human helps the robot and places the green block at its destination. The robot senses this, and looks at the policy derived for the sensed system state, resulting in a pick and place of the red block.

The program implementing such behavior is shown in Figure 7.8, along with its pTPN representation in Figure 7.9.

The user programs the specifications and a temporally consistent and chance-constrained execution policy is generated by the system. This is given as an input to *Enterprise*'s executive known as Pike [10]. Further, the demonstration shows that the system executes in real-time. The derived policy with timing constraints

```
def say(text):
  """Final message to the human."""
  return Episode(action='(say \"%s\")'%text)

def pick_and_place_block(prog,block,pick_loc,place_loc,manip,agent):
  """Picks a block and places it somewhere."""
  obj=block+'Component'
  return prog.sequence(
           say('Going to pick %s'%obj),
           Episode(action='(pick %s %s %s %s)'%(obj,manip,pick_loc,agent)),
           Episode(action='(place %s %s %s %s)'%(obj,manip,place_loc,agent)))

def observe_and_act(prog,blocks,manip,agent):
  """ Robot observes human and acts accordingly."""
  if len(blocks)>0:
      #Human helped with one of the blocks
      human_help=[observe_and_act(
                    prog,
                    [ob for ob in blocks if ob!=b],manip,agent) for b in blocks]
      #No help from the human
      no_human_help = prog.sequence(pick_and_place_block(prog,blocks[0],
                                                         blocks[0]+'Bin',
                                                         blocks[0]+'Target',
                                                         manip,agent),
                                    observe_and_act(prog,blocks[1:],manip,agent))
      #All episodes
      all_episodes = human_help+[no_human_help]

      #Observe each one of the blocks
      observations=blocks+['none']
      return prog.sequence(prog.observe({'name':'observe_human_%d'%(len(blocks)),
                                         'ctype':'uncontrollable',
                                         'domain':observations},
                                         *all_episodes))

  else:
    return say('All done!')

###### Control program starts here
blocks=['Red','Green']
agent='Baxter'
manip='BaxterRight'
prog = RMPyL(name='run()')
prog *= prog.sequence(say('Should I start?'),
                      prog.observe({'name':'observe_human_%d'%(len(blocks)),
                                    'ctype':'uncontrollable',
                                    'domain':['YES','NO']},
                                    observe_and_act(prog,blocks,manip,agent),
                                    say('All done!')))
```

Figure 7.8: Complete cRMPL program used to implement the collaborative manufacturing demonstration.

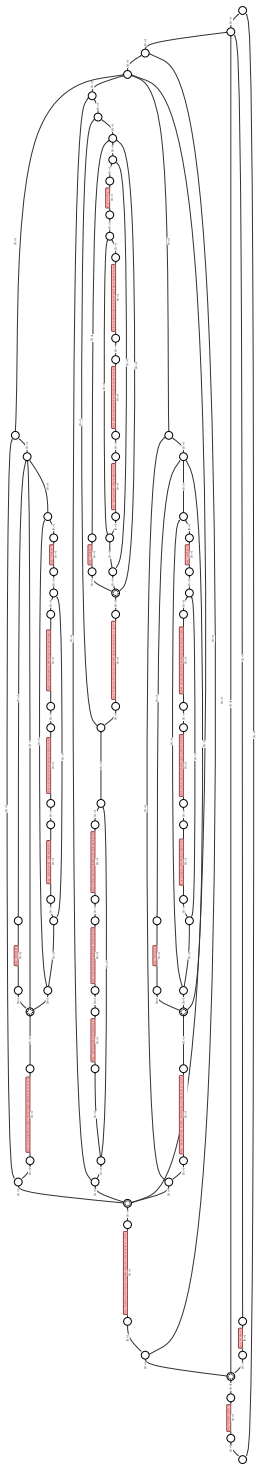


Figure 7.9: The policy for the Baxter demonstration.

is given in Figure 7.9. At every stage, the policy maps from the state of the world to the action the robot should perform in the next time step, including querying for further observations.

An intuition for understanding the policy would be to consider it as a sequence of IF statements. However, the policy also allows reasoning over timing constraints, otherwise difficult to capture with conventional control structures such as IF statements. Further, as shown by visual inspection, the actual policy for the problem is too tedious to actually encode by hand, even for expert users. Using our system, the user was able to describe the desired behavior in this case with simple statements specifying the end location of the blocks, and the risk-aware system automatically generated the policy. This allows ease of use by non-experts, one of the desired features of a deployable risk-aware autonomous planning and scheduling system.

7.2.1 Hybrid model learning in support of plan execution

When executing plans on real hardware, as in the manufacturing demonstration presented in this section, one no longer has full knowledge about the state of the environment, as it is usually the case in simulations. Instead, one must resort to different types of sensors, along with models of the environment, in order to be able to generate more human-understandable *predicates* such as "green block at its location" or "human has the red block on their hand", which are used during the planning and dispatching phases. In order to address this need, our group has developed LCARS, a predicate estimator based on Qualitative Spatial Reasoning (QSR) [43] and Probabilistic Hybrid Automata (PHA) [44, 45] models of the environment.

Similar to Section 7.1.1, one should ask the question of where one must acquire these models in order to perform state estimation of the surrounding environment and its agents. Towards that effort, we have proposed in [8] the first data-driven algorithm capable of learning PHA models directly from experimental data in the context of this project. The derivation of the algorithm is rather involved, but we show in our work that such PHA models can greatly enhance the performance of state estimators of complex systems, such as maneuvering aircraft and engineered systems with switched dynamics. We are currently in the process of applying the same techniques to LCARS, so that it can learn QSR models directly from experimental data, with little to no human supervision.

7.3 Automatic risk-aware program generation with RAO*

In this section, we show how RAO* (Chapter 5) is able to automatically generate cRMPL programs implementing safe, optimal behavior from a CC-POMDP model description. We demonstrate it in the aerial scout scenario depicted in Figures 7.10 and 7.11 featuring a Cessna 172P aircraft in the open-source flight simulator FlightGear.

The blue regions in Figure 7.10 represent areas of interest that might contain targets of various levels of importance. The confidence about the presence or absence of these targets in each one of the areas is given by a prior probability distribution, which is part of the model given to RAO*. The gray regions in Figure 7.10 are no-fly zones that the aircraft should avoid, and we use chance constraints to bound the probability of the aircraft inadvertently entering any one of them. When the aircraft flies over a region, it will discover a target with high probability should one be present, in which case it gains information reward. The risk-bounded path planning problem in this demonstration is solved by pSulu, the aforementioned chance-constrained path planner developed in our group.

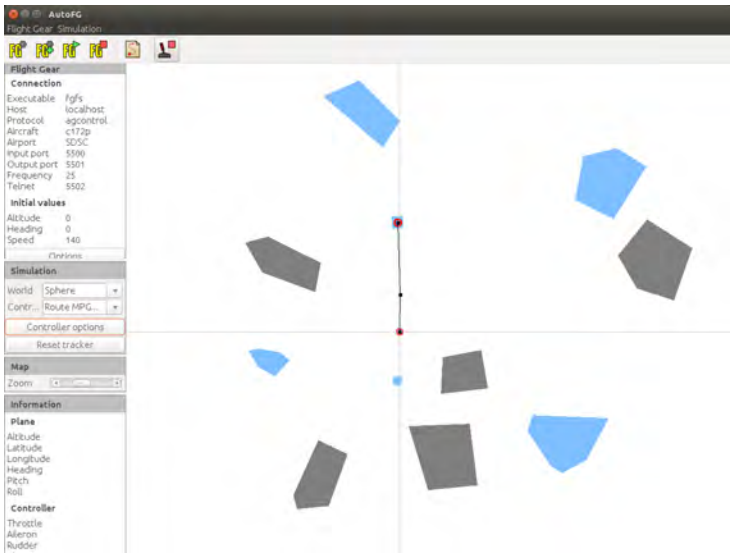


Figure 7.10: Mission map on AutoFG, the autopilot used to fly the Cessna in FlightGear. Blue regions are sites of interest that should be visited, time and risk permitting. Gray areas are no-fly zones that the aircraft should avoid.

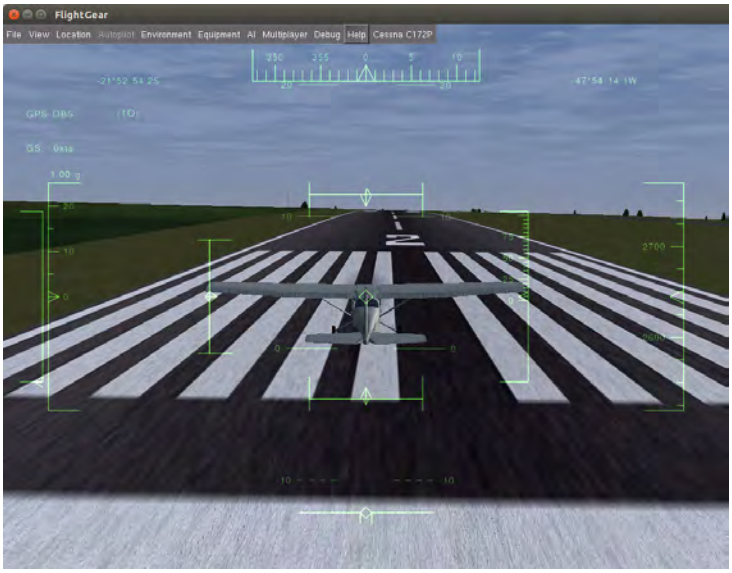


Figure 7.11: Screenshot of the FlightGear flight simulator, featuring a Cessna 172P aircraft.

Following the *Enterprise* architecture described in Chapter 2, cRMPL programs are compiled into a Probabilistic Temporal Plan Network format and dispatched by Pike. For illustration purposes, Figure 7.12 shows a pTPN corresponding to an over-constrained situation where the aircraft only has enough resources to visit a single site, while Figure 7.13 shows a snapshot of a cRMPL program generated by RAO* being dispatched by Pike within the *Enterprise* architecture. The full video can be found at `http://people.csail.mit.edu/psantana/public/videos/AFOSR/`.

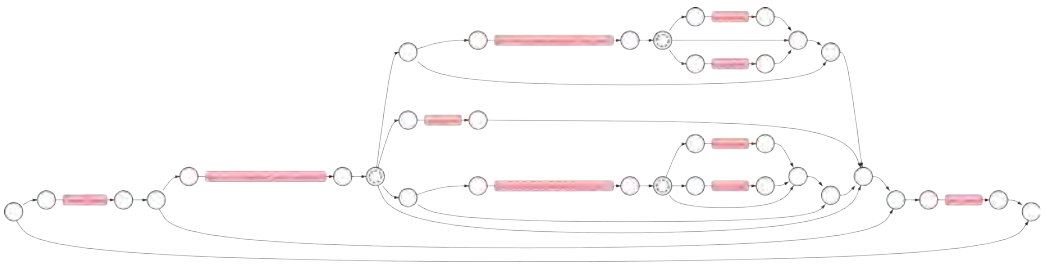


Figure 7.12: pTPN representation of an cRMPL program with enough resources to visit a single site.

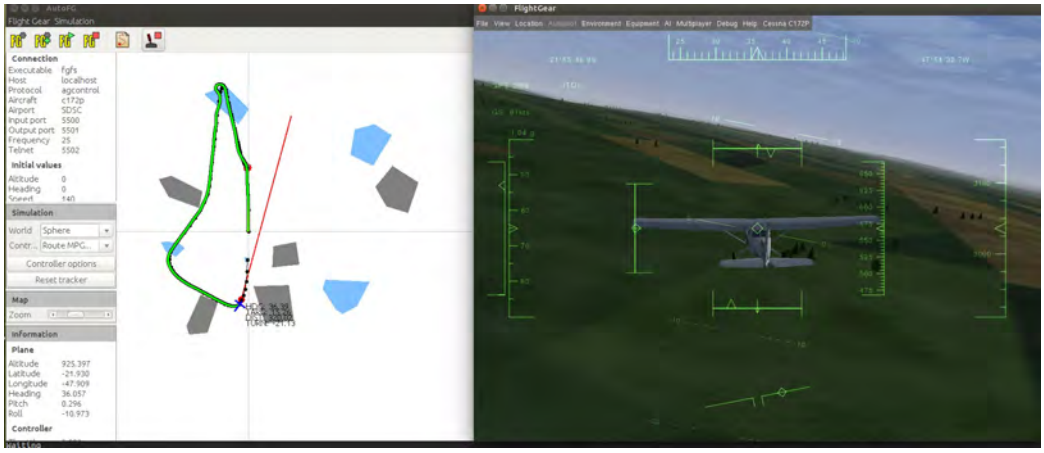


Figure 7.13: Snapshot of a cRMPL program generated by RAO* being dispatched by Pike within the *Enterprise* architecture.

A more detailed analysis of RAO*'s computational report can be found in Appendix A.

Chapter 8

Conclusions

This project developed *Enterprise*, a model-based programming and execution architecture allowing safe autonomous behavior to be specified at a natural level of abstraction, and contingent plans that are robust to environmental uncertainty to be synthesized, verified, and dispatched in real-time while ensuring good performance and hard guarantees on the risk of failure.

The chapters in this report and referenced publications describe our efforts in developing formal methods to accomplish the goals set forth in the desiderata, and our demonstrations, both in simulations and in real hardware, confirm that our chance-constrained architecture meets the desired goals of the project. All goals in our original Statement of Objectives were met according to plan or extended, and are briefly summarized below for convenience:

Objective 1 : create a chance-constrained, reactive model-based programming language (cRMPL) for specifying mission goals and risk levels.

Objective 2 : develop a risk-sensitive executive for cRMPL that uses its action model to plan an execution that maximizes expected utility, while respecting all chance constraints.

Objective 3 : validate cRMPL and *Enterprise* on a UAV mission (flight simulator or indoor quadcopters), spacecraft mission (MIT Spheres spacecraft) or a robot logistics mission.

The cRMPL language described in Chapter 3 and empirically demonstrated in Chapter 7 allows human operators to specify safe desired behavior at a high level of abstraction. It extends the previous version of RMPL by adding support to state

and temporal uncertainty, as well as safety guarantees in the form of chance constraints, while retaining RMPL's original capability of representing complex hierarchical compositions of plan episodes and their parallel coordination. Moreover, the choice of implementing cRMPL as a module of the general-purpose Python language allows it to be integrated within modern robotic frameworks, such as the *Robot Operating System* (ROS). The latter was a key feature that enabled cRMPL to control the Baxter robot in our robotic manufacturing demonstration in Chapter 7.

Chapters 4 through 6 and their accompanying peer-reviewed publications describe the different components required to accomplish the second objective. The formal methods and algorithms developed provide the user of *Enterprise* with a wide range of tools for checking contingent plans against safety specifications; generating optimal and safe contingent plans from a model description of the agent and its environment; and producing activity schedules that are robust not only to uncertainties in the timing of different actions, but also to the need of coordination of different autonomous agents under control. Our demonstrations in Chapter 7 confirm that these pieces, when integrated together within the *Enterprise* framework, allow autonomous agents to exhibit safe and optimal behavior while operating in partially-known environments. Moreover, we provide references and intuitions related to our work in learning task and environmental models from experimental and simulated data, which was carried out in support of enabling *Enterprise* to be demonstrated in real-world settings.

Concerning the experimental validation in the third objective, we demonstrated *Enterprise* both in simulated UAV missions (FlightGear), as well as using real quadcopters in the undergraduate course described in Chapter 2. We have also shown its usefulness in a robotic manufacturing test bed purchased under this contract. The *Enterprise* architecture is essentially unchanged among all sections in Chapter 7, as well as Chapter 2: the planner and scheduler produce an execution policy, which is dispatched in real-time by Pike. This architecture is thus transferable and shown to be easy to use by non-experts, having been demonstrated on vastly different hardware and software.

Appendix A

RAO*: an Algorithm for Chance-Constrained POMDP's

This appendix contains a detailed description and experimental evaluation of Risk-aware AO* (RAO*), which was introduced in Chapter 5 in the context of model-based generation of cRMPL programs. It is currently under review for publication at the 30^{th} Conference on Artificial Intelligence (AAAI16).

RAO*: an Algorithm for Chance-Constrained POMDP's

Pedro Santana*, Sylvie Thiébaux+, Brian Williams*
*Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, MERS
32 Vassar St. Room 32-224, Cambridge, MA 02139, {psantana,williams}@mit.edu
+The Australian National University & NICTA
Canberra ACT 0200, Australia, Sylvie.Thiebaux@anu.edu.au

Abstract

Autonomous agents operating in partially observable stochastic environments often face the problem of optimizing expected performance while bounding the risk of violating safety constraints. Such problems can be modeled as chance-constrained POMDP's (CC-POMDP's). Our first contribution is a systematic derivation of execution risk in POMDP domains, which improves upon how chance constraints are handled in the constrained POMDP literature. Second, we present RAO*, a heuristic forward search algorithm producing optimal, deterministic, finite-horizon policies for CC-POMDP's. In addition to the utility heuristic, RAO* leverages an admissible execution risk heuristic to quickly detect and prune overly-risky policy branches. Third, we demonstrate the usefulness of RAO* in two challenging domains of practical interest: power supply restoration and autonomous science agents.

1 Introduction

Partially Observable Markov Decision Processes (POMDPs) (Smallwood and Sondik 1973) have become one of the most popular frameworks for optimal planning under actuator and sensor uncertainty, where POMDP solvers find policies that maximize some measure of expected utility (Kaelbling, Littman, and Cassandra 1998; Silver and Veness 2010).

In many application domains, however, performance is not enough. Critical missions in real-world scenarios require agents to develop a keen sensitivity to risk, which needs to be traded-off against utility. For instance, a search and rescue UAV should maximize the value of the information gathered, subject to safety constraints such as avoiding dangerous areas and keeping sufficient battery levels. In these domains, autonomous agents should seek to optimize expected reward while remaining safe by deliberately keeping the probability of violating one or more constraints within acceptable levels. A bound on the probability of violating constraints is called a *chance constraint* (Birge and Louveaux 1997). Unsurprisingly, attempting to model chance constraints as negative rewards leads to models that are oversensitive to the particular penalty value chosen, and to policies that are overly risk-averse or overly risk-taking (Undurti and How 2010). Therefore, to accommodate the type of scenarios exemplified above, new models and algorithms for *constrained* MDPs have started to emerge, which handle chance constraints explicitly.

Research has mostly focused on fully observable constrained MDPs, for which non-trivial theoretical properties are known (Altman 1999; Feinberg and Shwarz 1995). Existing algorithms cover an interesting spectrum of chance constraints over secondary objectives or even execution paths, e.g., (Dolgov and Durfee 2005; Hou, Yeoh, and Varakantham 2014; Teichteil-Königsbuch 2012). For constrained POMDPs (C-POMDP's), the state of the art is less mature. It includes a few suboptimal or approximate methods based on extensions of dynamic programming (Isom, Meyn, and Braatz 2008), point-based value iteration (Kim et al. 2011), approximate linear programming (Poupart et al. 2015), or on-line search (Undurti and How 2010). Moreover, as we later show, the modeling of chance constraints through unit costs in the C-POMDP literature has a number of shortcomings.

Our first contribution is a systematic derivation of the *execution risk* in POMDP domains, and how it can be used to enforce different types of chance constraints. Second, we present Risk-bounded AO* (RAO*), a new algorithm for solving chance-constrained POMDPs (CC-POMDPs) that harnesses the power of heuristic forward search in belief space (Washington 1996; Bonet and Geffner 2000; Szer, Charpillet, and Zilberstein 2005; Bonet and Geffner 2009). Similar to AO* (Nilsson 1982), RAO* guides the search towards promising policies w.r.t. reward using an admissible heuristic. Third, RAO* leverages a second admissible heuristic to derive and propagate execution risk upper bounds at each search node, allowing it to identify and prune overly risky paths as the search proceeds. Last, we demonstrate the usefulness of RAO* in two risk-sensitive domains of practical interest: automated power supply restoration (PSR) and autonomous science agents (SA).

RAO* returns policies that maximize the expected cumulative reward among the set of deterministic, finite-horizon policies satisfying the chance constraints. Even though optimal policies for CC-(PO)MDPs may, in general, require some limited amount of randomization (Altman 1999), we follow Dolgov and Durfee (2005) in deliberately developing an approach restricted to deterministic policies. This is mo-

tivated by the fact that users rarely trust stochastic decisions when dealing with safety-critical applications.

The paper is organized as follows. Section 2 formulates the type of CC-POMDPs we consider, and details how RAO* computes execution risks and propagates risk bounds forward. Next, Section 3 discusses shortcomings related to the treatment of chance constraints in the C-POMDP literature. Section 4 presents the RAO* algorithm, followed by our experiments in Section 5, and conclusions in Section 6.

2 Problem formulation

When the true state of the system is hidden, one can only maintain a probability distribution (a.k.a. belief state) over the possible states of the system at any given point in time. Many applications in which an agent is trying to act under uncertainty while optimizing some measure of performance can be adequately framed as instances of Partially Observable Markov Decision Processes (POMDP) (Smallwood and Sondik 1973). Here, we focus on the case where there is a finite policy execution horizon h, after which the system performs a deterministic transition to an absorbing state.

Definition 1 (Finite-horizon POMDP)**.** *A FH-POMDP is a tuple* $H =< \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, b_0, h >$*, where* $\mathcal{S}$ *is a set of states;* $\mathcal{A}$ *is a set of actions;* $\mathcal{O}$ *is a set of observations;* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ *is a stochastic state transition function;* $O : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ *is a stochastic observation function;* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is a reward function;* b_0 *is the initial belief state; and* h *is the execution horizon.*

A solution to an FH-POMDP is a mapping $\pi : \mathcal{B} \to \mathcal{A}$ from beliefs to actions, called a *policy*. An optimal policy π^* is such that

$$\pi^* = \arg_\pi \max \mathbb{E}\left[\sum_{t=0}^{h} R(s_t, a_t)\Big|\pi\right]. \tag{1}$$

In this work, we focus on the particular case of discrete $\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$, and deterministic optimal policies.

Let $\hat{b}_k : \mathcal{S} \to [0, 1]$ denote the *posterior* belief state at the k-th time step. A belief state at time $k+1$ that only incorporates information about the most recent action a_k is called a *prior* belief state and denoted by $\bar{b}(s_{k+1}|a_k)$. If, besides a_k, the belief state also incorporates knowledge from the most recent observation o_{k+1}, we call it a *posterior* belief state and denote it by $\hat{b}(s_{k+1}|a_k, o_{k+1})$. These beliefs can be recursively computed as follows:

$$\bar{b}(s_{k+1}|a_k) = \Pr(s_{k+1}|\hat{b}_k, a_k) = \sum_{s_k} T(s_k, a_k, s_{k+1})\hat{b}(s_k) \tag{2}$$

$$\hat{b}(s_{k+1}|a_k, o_{k+1}) = \Pr(s_{k+1}|\hat{b}_k, a_k, o_{k+1}),$$
$$= \frac{1}{\eta} O(s_{k+1}, o_{k+1})\bar{b}(s_{k+1}|a_k), \tag{3}$$

where T and O are from Definition 1, and

$$\eta = \Pr(o_{k+1}|a_k, b_k) = \sum_{s_{k+1}} O(s_{k+1}, o_{k+1})\bar{b}(s_{k+1}|a_k) \tag{4}$$

is the probability of collecting some observation o_{k+1} after executing action a_k at a belief state b_k.

In addition to optimizing performance, the next session shows how one can enforce safety in FH-POMDPs by means of chance constraints.

2.1 Computing risk

A chance constraint consists of a bound Δ on the probability (chance) of some event happening during policy execution. Following (Ono, Kuwata, and Balaram 2012), we define this event as a sequence of states $s_{0:h} = s_0, s_1, \ldots, s_h$ of a FH-POMDP H violating one or more constraints in a set C. Let p (for "path") denote a sequence of states $p = s_{0:h}$, and let $c_v(p) \in \{0, 1\}$ be an indicator function such that $c_v(p) = 1$ iff one or more states in p violate constraints in C. The latter implies that states encompass all the information required to evaluate constraints. With this notation, we can write the chance constraint as

$$\mathbb{E}_p\left(\, c_v(p)|\, H, \pi\right) \leq \Delta, \tag{5}$$

One should notice that we make no assumptions about constraint violations producing observable outcomes, such as causing execution to halt.

Our approach for incorporating chance constraints into FH-POMDP's extends that of (Ono and Williams 2008; Ono, Kuwata, and Balaram 2012) to partially observable planning domains. We would like to be able to compute increasingly better approximations of (5) with admissibility guarantees, so as to be able to quickly detect policies that are guaranteed to violate (5). For that purpose, let b_k be a belief state over s_k, and Sa_k be a Bernoulli random variable denoting whether the system has not violated any constraints (is "safe") at time k. We define

$$er(b_k, C|\pi) = 1 - \Pr\left(\bigwedge_{i=k}^{h} Sa_i \,\middle|\, b_k, \pi\right) \tag{6}$$

as the *execution risk* of policy π, as measured from b_k. The probability term in (6) can be written as

$$\Pr\left(\bigwedge_{i=k}^{h} Sa_i \,\middle|\, b_k, \pi\right)$$
$$= \Pr\left(\bigwedge_{i=k+1}^{h} Sa_i \,\middle|\, Sa_k, b_k, \pi\right)\Pr(Sa_k|b_k, \pi), \tag{7}$$

where $\Pr(Sa_k|b_k, \pi)$ is the probability of the system not being in a constraint-violating path at the k-th time step. Since b_k is given, $\Pr(Sa_k|b_k, \pi)$ can be computed as

$$\Pr(Sa_k|b_k, \pi) = 1 - \sum_{s_k \in S} b(s_k)c_v(s_k, C) = 1 - r_b(b_k, C), \tag{8}$$

where $r_b(b_k, C)$ is called the *risk* at the k-th step. Note that $c_v(s_k, C) = 1$ iff s_k or any of its ancestor states violate constraints in C. In situations where the particular set of constraints C is not important, we will use the shorthand notation $r_b(b_k)$ and $er(b_k|\pi)$. The second probability term in (7) can be written as

$$\Pr\left(\bigwedge_{i=k+1}^{h} Sa_i \,\middle|\, Sa_k, b_k, \pi\right)$$
$$= \sum_{b_{k+1}} \Pr\left(\bigwedge_{i=k+1}^{h} Sa_i \,\middle|\, b_{k+1}, \pi\right)\Pr(b_{k+1}|Sa_k, b_k, \pi),$$
$$= \sum_{b_{k+1}} (1 - er(b_{k+1}|\pi))\Pr(b_{k+1}|Sa_k, b_k, \pi). \tag{9}$$

The summation in (9) is over belief states at time $k+1$. These, in turn, are determined by (3), with $a_k = \pi(b_k)$ and some corresponding observation o_{k+1}. Therefore, we have $\Pr(b_{k+1}|Sa_k, b_k, \pi) = \Pr(o_{k+1}|Sa_k, \pi(b_k), b_k)$. For the purpose of computing the RHS of the last equation, it is useful to define *safe prior* belief as

$$\bar{b}^{sa}(s_{k+1}|a_k) = \Pr(s_{k+1}|Sa_k, a_k, b_k), \\ = \frac{\sum_{s_k: c_v(s_k, C)=0} T(s_k, a_k, s_{k+1}) b(s_k)}{1 - r_b(b_k)}, \quad (10)$$

With (10), we can define

$$\Pr^{sa}(o_{k+1}|a_k, b_k) = \Pr(o_{k+1}|Sa_k, a_k, b_k), \\ = \sum_{s_{k+1}} O(s_{k+1}, o_{k+1}) \bar{b}^{sa}(s_{k+1}|a_k), \quad (11)$$

which is the distribution over observations at time $k+1$, assuming that the system was in a non-violating path at time k. Combining (6), (8), (9), and (11), we get the recursion

$$er(b_k|\pi) = r_b(b_k) \\ + (1 - r_b(b_k)) \sum_{o_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}|\pi), \quad (12)$$

which is key to RAO*. If b_k is terminal, (12) simplifies to $er(b_k|\pi) = r_b(b_k)$. Note that (12) uses (11), rather than (4), to compute execution risk. We can now use the execution risk to express the chance constraint (5) in our definition of a chance-constrained POMDP (CC-POMDP).

Definition 2 (Chance-constrained POMDP)**.** *A CC-POMDP is a tuple $< H, \mathcal{C}, \Delta >$, where H is a FH-POMDP; $\mathcal{C}$ is a set of constraints defined over $\mathcal{S}$; and $\Delta = [\Delta^1, \ldots, \Delta^q]$ is a vector of probabilities for q chance constraints*

$$er(b_0, C^i|\pi) \leq \Delta^i, C^i \in 2^{\mathcal{C}}, i = 1, 2, \ldots, q. \quad (13)$$

The chance constraint in (13) bounds the probability of constraint violation over the whole policy execution. Alternative forms of chance constraints entailing safer behavior are discussed in Section 2.3. Our approach for finding optimal, deterministic, chance-constrained solutions for CC-POMDP's in this setting is explained in Section 4.

2.2 Propagating risk bounds forward

The approach for computing risk in (12) does so "backwards", i.e., risk propagation happens from terminal states to the root of the search tree. However, since we propose computing policies for chance-constrained POMDP's by means of heuristic forward search, one should seek to propagate the risk bound in (13) forward so as to be able to quickly detect that the current best policy is too risky.

Let $0 \leq \tilde{\Delta}_k \leq 1$ be the bound on execution risk for node b_k. From (12), we get

$$r_b(b_k) + (1 - r_b(b_k)) \sum_{o_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}|\pi) \leq \tilde{\Delta}_k. \quad (14)$$

Let o'_{k+1} such that $\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k) \neq 0$ be the observation associated with the child b'_{k+1} of b_k. From (14), we get

$$er(b'_{k+1}|\pi) \leq \frac{1}{\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)} \left(\frac{\tilde{\Delta}_k - r_b(b_k)}{1 - r_b(b_k)} \right. \\ \left. - \sum_{o_{k+1} \neq o'_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}|\pi) \right). \quad (15)$$

The existence of (15) requires $r_b(b_k) < 1$ and $\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k) \neq 0$ whenever $\Pr(o'_{k+1}|\pi(b_k), b_k) \neq 0$. Lemma 1 shows that these conditions are equivalent.

Lemma 1. *One observes $\Pr^{sa}(o_{k+1}|\pi(b_k), b_k) = 0$ and $\Pr(o_{k+1}|\pi(b_k), b_k) \neq 0$ if, and only if, $r_b(b_k) = 1$.*

Proof. $\Leftarrow$: if $r_b(b_k) = 1$, we conclude from (8) that $c_v(s_k, C) = 1, \forall s_k$. Hence, all elements in (10) and, consequently, (11) will have probability 0.

$\Rightarrow$: from Bayes' rule, we have

$$\Pr(Sa_k|o_{k+1}, a_k, b_k) = \frac{\Pr^{sa}(o_{k+1}|a_k, b_k)(1 - r_b(b_k))}{\Pr(o_{k+1}|a_k, b_k)} = 0$$

Hence, we conclude that $\Pr(\neg Sa_k|o_{k+1}, a_k, b_k) = 1$, i.e., the system is guaranteed to be in a constraint-violating path at time k, yielding $r_b(b_k) = 1$.

□

The execution risk of nodes whose parents have $r_b(b_k) = 1$ is irrelevant, as shown by (12). Therefore, it only makes sense to propagate risk bounds in cases where $r_b(b_k) < 1$.

One difficulty associated with (15) is that it depends on the execution risk of all siblings of b'_{k+1}, which cannot be computed exactly until terminal nodes are reached. Therefore, one must approximate (15) in order to render it computable during forward search.

We can easily define a necessary condition for feasibility of a chance constraint at a search node by means of an admissible execution risk heuristic $h_{er}(b_{k+1}|\pi) \leq er(b_{k+1}|\pi)$. Combining $h_{er}(\cdot)$ and (15) provides us with a necessary condition

$$er(b'_{k+1}|\pi) \leq \frac{1}{\Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)} \left(\frac{\tilde{\Delta}_k - r_b(b_k)}{1 - r_b(b_k)} \right. \\ \left. - \sum_{o_{k+1} \neq o'_{k+1}} \Pr^{sa}(o_{k+1}|\pi(b_k), b_k) h_{er}(b_{k+1}|\pi) \right). \quad (16)$$

Since $h_{er}(b_{k+1}|\pi)$ computes a lower bound on the execution risk, we conclude that (16) gives an upper bound for the true execution risk bound in (15). The simplest possible heuristic is $h_{er}(b_{k+1}|\pi) = 0, \forall b_{k+1}$, which assumes that it is absolutely safe to continue executing policy π beyond b_k. Moreover, from the non-negativity of the terms in (12), we see that another possible choice of a lower bound is $h_{er}(b_{k+1}|\pi) = r_b(b_{k+1})$, which is guaranteed to be an improvement over the previous heuristic, for it incorporates additional information about the risk of failure at that belief state. However, it is still a myopic risk estimate, given that it ignores the execution risk for nodes beyond b_{k+1}. All these bounds can be compute forward, starting with $\tilde{\Delta}_0 = \Delta$.

2.3 Enforcing safe behavior at all times

Enforcing (13) bounds the probability of constraint violation over total policy executions, but (15) shows that unlikely policy branches can be allowed risks close or equal to 1 if that will help improve the objective, giving rise to a "daredevil" attitude. Since this might not be the desired risk-aware behavior, a straightforward way of achieving higher levels of safety is to depart from the chance constraints in (13) and, instead, impose a set of chance constraints of the form

$$er(b_k, C^i|\pi) \leq \Delta^i, \forall i, b_k \text{ s.t. } b_k \text{ is nonterminal.} \quad (17)$$

Intuitively, (17) tells the autonomous agent to "remain safe at all times", whereas the message conveyed by (13) is "stay safe overall". It should be clear that (17)⇒(13), so (17) necessarily generates safer policies than (13), but also more conservative in terms of utility. Another possibility is to follow (Ono, Kuwata, and Balaram 2012) and impose

$$\sum_{k=0}^{h} r_b(b_k, C^i) \leq \Delta^i, \forall i, \quad (18)$$

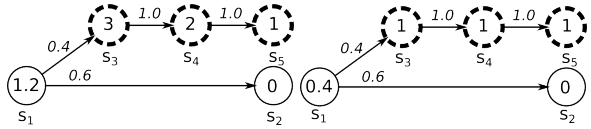
which is a sufficient condition for (13) based on Boole's inequality. One can show that (18)⇒(17), so enforcing (18) will lead to policies that are at least as conservative as (17).

3 Relation to constrained POMDP's

Alternative approaches for chance-constrained POMDP planning have been presented in (Undurti and How 2010) and (Poupart et al. 2015), where the authors propose algorithms for solving constrained POMDP's (C-POMDP's). They argue that chance constraints can be modeled within the C-POMDP framework by assigning unit costs to states violating constraints, 0 to others, and proceeding with calculations as usual.

There are two main shortcomings associated with the use of unit costs to deal with chance constraints. First, it only yields correct measures of execution risk in the particular case where constraint violations cause policy execution to terminate. If that is not the case, incorrect probability values can be attained, as shown in the simple example in Figure 1. Second, assuming that constraint violations cause execution to cease has a strong impact on belief state computations. The key insight here is that assuming that constraint violations cause execution to halt provides the system with an invaluable observation: at each non-terminal belief state, the risk $r_b(b_k, C)$ in (8) must be 0. The reason for that is simple: (*constraint violation* ⇒ *terminal belief*) ⇔ (*non-terminal belief* ⇒ *no constraint violation*).

Assuming that policy execution terminates at constraint violations is reasonable when undesirable states are destructive, e.g., the agent is destroyed after crashing against an obstacle. Nevertheless, it is rather limiting in terms of expressiveness, since there are application domains where undesirable states can be "benign". For instance, in the power supply restoration domain described in the experimental section, connecting faults to generators is undesirable and we want to limit the probability of this event. However, it does not destroy the network. In fact, it might be the only way to significantly reduce the uncertainty about the location of a load fault, therefore allowing for a larger amount of power to be restored to the system.



(a) Incorrect execution risks computed using unit costs. (b) Correct execution risks computed according to (12).

Figure 1: Modeling chance constraints via unit costs may yield incorrect results when constraint-violating states (dashed outline) are not terminal. Numbers within states are constraint violation probabilities. Numbers over arrows are probabilities for a non-deterministic action.

4 Solving CC-POMDP's through RAO*

In this section, we introduce the Risk-bounded AO* algorithm (RAO*) for constructing risk-bounded policies for CC-POMDP's. RAO* is based on heuristic forward search in the space of belief states. The motivation for this is simple: given an initial belief state and limited resources (including time), the number of *reachable* belief states from a set of initial conditions is usually a very small fraction of the total number of possible belief states. Another reason is that there might not be a clear concept of a "goal state", which makes it hard to perform goal regression.

Similar to AO* in fully observable domains, RAO* (Algorithm 1) explores its search space of belief states from the initial belief b_0 by incrementally constructing a hypergraph G called the *explicit* hypergraph. Each node in G represents a belief state, and a hyperedge is a compact representation of the process of taking an action and receiving any of a number of possible observations. Each node in G is associated with the Q value

$$Q(b_k, a_k) = \sum_{s_k} R(s_k, a_k) b(s_k) + \sum_{o_{k+1}} \Pr(o_{k+1}|a_k, b_k) Q^*(b_{k+1}) \quad (19)$$

representing the expected, cumulative reward of taking action a_k at some belief state b_k. The first term corresponds to the expected current reward, while the second term is the expected reward obtained by following the optimal deterministic policy π^*, i.e., $Q^*(b_{k+1}) = Q(b_{k+1}, \pi^*(b_{k+1}))$. Given an admissible estimate $h_Q(b_{k+1})$ of $Q^*(b_{k+1})$, we select actions for the current estimate $\hat{\pi}$ of π^* according to

$$\hat{\pi}(b_k) = \arg\max_{a_k} \hat{Q}(b_k, a_k), \quad (20)$$

where $\hat{Q}(b_k, a_k)$ is the same as (19) with $Q^*(b_{k+1})$ replaced by $h_Q(b_{k+1})$. The portion of G corresponding to the current estimate $\hat{\pi}$ of π^* is called the *greedy* graph, for it uses an admissible heuristic estimate $h_Q(b_k, a_k)$ of $Q^*(b_{k+1})$ to explore the most promising areas of G first.

The most important differences between AO* and RAO* lie in Algorithms 2 and 3. First, since RAO* deals with partially observable domains, node expansion in Algorithm 2

Algorithm 1 RAO*

Input: CC-POMDP H, initial belief b_0.
Output: Optimal policy π mapping beliefs to actions.
1: Explicit graph G and policy π initially consist of b_0.
2: **while** π has some nonterminal leaf node **do**
3: $n, G \leftarrow$ expand-policy(G, π)
4: $\pi \leftarrow$ update-policy(n, G, π)
5: **return** π.

Algorithm 2 expand-policy

Input: Explicit graph G, policy π.
Output: Expanded explicit G', expanded leaf node n.
1: $G' \leftarrow G$, $n \leftarrow$ choose-promising-leaf(G, π)
2: **for** each action a available at n **do**
3: $ch \leftarrow$ use (2), (3), (4) to expand children of (n, a).
4: $\forall c \in ch$, use (8), (11), (12), and (19) with admissible heuristics to estimate Q^* and er.
5: $\forall c \in ch$, use (16) to compute exec. risk bounds
6: **if** no $c \in ch$ violates its risk bound **then**
7: $G' \leftarrow$ add hyperedge $[(n, a) \rightarrow ch]$
8: **if** no action added to n **then** mark n as terminal.
9: **return** G', n.

involves full Bayesian prediction and update steps, as opposed to a simple branching using the state transition function T. In addition, RAO* leverages the heuristic estimates of execution risk explained in Section 2.2 in order to perform early pruning of actions that introduce child belief nodes that are guaranteed to violate the chance constraint. The same process is also observed during policy update in Algorithm 3, in which heuristic estimates of the execution risk are used to prevent RAO* to keep choosing actions that are promising in terms of heuristic value, but can be proven to violate the chance constraint at an early stage.

The proofs of soundness, completeness, and optimality for RAO* are given in Lemma 2 and Theorem 1.

Lemma 2. *Risk-based pruning of actions in Algorithms 2 (line 6) and 3 (line 7) is sound.*

Proof. The RHS of (15) is the true execution risk bound for $er(b'_{k+1}|\pi)$. The execution risk bound on the RHS of (16) is an upper bound for the bound in (15), since we replace $er(b_{k+1}|\pi)$ for the siblings of b'_{k+1} by admissible estimates (lower bounds) $h_{er}(b_{k+1}|\pi)$. In the aforementioned pruning steps, we compare $h_{er}(b'_{k+1}|\pi)$, a lower bound on the true value $er(b'_{k+1}|\pi)$, to the upper bound (16). Verifying $h_{er}(b'_{k+1}|\pi) > (16)$ is sufficient to establish $er(b'_{k+1}|\pi) > (15)$, i.e., action a currently under consideration is guaranteed to violate the chance constraint. □

Algorithm 3 update-policy

Input: Expanded n, explicit graph G, policy π.
Output: Updated policy π'.
1: $Z \leftarrow$ set containing n and its ancestors reachable by π.
2: **while** $Z \neq \emptyset$ **do**
3: $n \leftarrow$ remove(Z) node n with no descendant in Z.
4: **while** there are actions to be chosen at n **do**
5: $a \leftarrow$ next best action at n according to (20) satisfying exec. risk bound.
6: Propagate execution risk bound of n to the children of the hyperedge (n, a)
7: **if** no children violates its exec. risk bound **then**
8: $\pi(n) \leftarrow a$; **break**
9: **if** no action was selected at n **then** mark n as terminal

Theorem 1. *RAO* is complete and produces the optimal deterministic, finite-horizon policies meeting the chance constraints.*

Proof. A CC-POMDP, as described in Definition 2, has a finite number of policy branches, and Lemma 2 shows that RAO* only prunes policy branches that are guaranteed not to be part of any chance-constrained solution. Therefore, if no chance-constrained policy exists, RAO* will eventually return an empty policy.

Concerning the optimality of RAO* with respect to the utility function, it follows from the admissibility of $h_Q(b_k, a_k)$ in (20) and the optimality guarantee of AO*. □

5 Experiments

This section provides empirical evidence of the usefulness and general applicability of CC-POMDP's as modeling tool for risk-sensitive applications, and shows how RAO* performs when computing risk-bounded policies in two challenging domains of practical interest: power supply restoration (PSR) (Thiébaux and Cordier 2001) and automated planning for science agents (SA) (Benazera et al. 2005). All models and RAO* were implemented in Python and ran on an Intel Core i7-2630QM CPU with 8GB of RAM.

In the *PSR* domain (Thiébaux and Cordier 2001), the objective is to reconfigure a faulty power network by switching lines on or off so as to resupply as many customers as possible. One of the safety constraints is to keep faults isolated at all times, to avoid endangering people and enlarging the set of areas left without power. However, fault locations are hidden, and more information cannot be obtained without taking the risk of resupplying a fault. Therefore, the chance constraint is used to limit the probability of connecting power generators to faulty buses. Our experiments focused on the semi-rural network from (Thiébaux and Cordier 2001), which was significantly beyond the reach of (Bonet and Thiébaux 2003) even for single faults. In our experiments, there were always circuit breakers at each generator, plus different numbers of additional circuit breakers depending on the experiment. Observations correspond to circuit breakers being open or closed, and actions to opening and closing switches. The PSR domain is strongly combinatorial, with $|\mathcal{S}| = 2^{61}; |\mathcal{A}| = 68, |\mathcal{O}| = 32$.

Our SA domain is based on the planetary rover scenario described in (Benazera et al. 2005). Starting from some initial position in a map with obstacles, the science agent may visit four different sites on the map, each of which could contain new discoveries with probability based on a prior belief. If the agent visits a location that contains new discoveries, it will find it with high probability. The agent's position is uncertain, so there is always a non-zero risk of

collision when the agent is traveling between locations. The agent is required to finish its mission at a relay station, where it can communicate with an orbiting satellite and transmit its findings. Since the satellite moves, there is a limited time window for the agent to gather as much information as possible and arrive at the relay station. Moreover, we assume the duration of each traversal to be uncontrollable, but bounded. In this domain, we use a single chance constraint to ensure that the event "arrives at the relay location on time" happens with probability at least $1 - \Delta$. The SA domain has size $|\mathcal{S}| = 6144; |\mathcal{A}| = 34, |\mathcal{O}| = 10$.

We evaluated the performance of RAO* in both domains under various conditions, and the results are summarized in Tables 1 (higher utility is better) and 2 (lower cost is better). It is worthwhile to mention that constraint violations in PSR *do not cause execution to terminate*, and the same is true for scheduling violations in SA. The only type of terminal constraint violation are collisions in SA, and RAO* makes proper use of this extra bit of information to update its beliefs. Therefore, PSR and SA are examples of risk-sensitive domains which can be appropriately modeled as CC-POMDP's, but not as C-POMDP's with unit costs. The heuristics used were straightforward: for the execution risk, we used the admissible heuristic $h_{er}(b_k|\pi) = r_b(b_k)$ in both domains. For Q values, the heuristic for each state in PSR consisted in the final penalty incurred if only its faulty nodes were not resupplied, while in SA it was the sum of the utilities of all non-visited discoveries.

As expected, both tables show that increasing the maximum amount of risk Δ allowed during execution can only improve the policy's objective. The improvement is not monotonic, though. The impact of the chance constraint on the objective is discontinuous on Δ when only deterministic policies are considered, since one cannot randomly select between two actions in order to achieve a continuous interpolation between risk levels. Being able to compute increasingly better approximations of a policy's execution risk, combined with forward propagation of risk bounds, also allow RAO* to converge faster by quickly pruning candidate policies that are guaranteed to violate the chance constraint. This can be clearly observed in Table 2 when we move from $\Delta = 0.5$ to $\Delta = 1.0$ (no chance constraint).

Another important aspect is the impact of sensor information on the performance of RAO*. Adding more sources of sensing information increases the branching on the search hypergraph used by RAO*, so one could expect performance to degrade. However, that is not necessarily the case, as shown by the left and right numbers in the cells of Table 2. By adding more sensors to the power network, RAO* can more quickly reduce the size of its belief states, therefore leading to a reduced number of states evaluated during search. Another benefit of reduced belief states is that RAO* can more effectively reroute energy in the network within the given risk bound, leading to lower execution costs.

Finally, we wanted to investigate how well a C-POMDP approach would perform in these domains relative to a CC-POMDP. Following the literature, we made the additional assumption that execution halts at all constraint violations, and assigned unit terminal costs to those search nodes. Results on two example instances of PSR and SA domains were the following: I) in SA, C-POMDP and CC-POMDP both attained an utility of 29.454; II) in PSR, C-POMDP reached a final cost of 53.330, while CC-POMDP attained 36.509. The chance constraints were always identical for C-POMDP and CC-POMDP. First, one should notice that both models had the same performance in the SA domain, which is in agreement with the claim that they coincide in the particular case were all constraint violations are terminal. The same, however, clearly does not hold in the PSR domain, where the C-POMDP model had significantly worse performance than its corresponding CC-POMDP with the exact same parameters. Assuming that constraint violations are terminal in order to model them as costs greatly restricts the space of potential solution policies in domains with non-destructive constraint violations, leading to conservatism. A CC-POMDP formulation, on the other hand, can potentially attain significantly better performance while offering the same safety guarantee.

Window[s]	Δ	Time[s]	Nodes	States	Utility
20	0.05	1.30	1	32	0.000
30	0.01	1.32	1	32	0.000
30	0.05	49.35	83	578	29.168
40	0.002	9.92	15	164	21.958
40	0.01	44.86	75	551	29.433
40	0.05	38.79	65	443	29.433
100	0.002	95.23	127	1220	24.970
100	0.01	184.80	161	1247	29.454
100	0.05	174.90	151	1151	29.454

Table 1: SA results for various time windows and risk levels.

Δ	Time[s]	Nodes	States	Cost
0	0.025/0.013	1.57/1.29	5.86/2.71	45.0/30.0
.5	0.059/0.014	3.43/1.29	10.71/2.71	44.18/30.0
1	2.256/0.165	69.3/11.14	260.4/23.43	30.54/22.89
0	0.078/0.043	2.0/1.67	18.0/8.3	84.0/63.0
.5	0.157/0.014	3.0/1.29	27.0/2.71	84.0/30.0
1	32.78/0.28	248.7/5.67	1340/32.33	77.12/57.03
0	1.122/0.093	7.0/2.0	189.0/12.0	126.0/94.50
.5	0.613/0.26	4.5/4.5	121.5/34.5	126.0/94.50
1	123.9/51.36	481.5/480	8590.5/2648	117.6/80.89

Table 2: PSR results for various numbers of faults (#) and risk levels. Top: avg. of 7 single faults. Middle: avg. of 3 double faults. Bottom: avg. of 2 triple faults. Left (right) numbers correspond to 12 (16) network sensors.

6 Conclusions

We have presented RAO*, an algorithm for optimally solving CC-POMDP's. By combining the advantages of AO* in the belief space with forward propagation of risk upper bounds, RAO* is able to solve challenging risk-sensitive planning problems of practical interest and size. Our agenda for future work includes generalizing the algorithm to move away from the finite horizon setting, as well as more general chance constraints, including temporal logic path constraints (Teichteil-Königsbuch 2012).

References

Altman, E. 1999. *Constrained Markov Decision Processes*, volume 7. CRC Press.

Benazera, E.; Brafman, R.; Meuleau, N.; Hansen, E. A.; et al. 2005. Planning with continuous resources in stochastic domains. In *International Joint Conference on Artificial Intelligence*, volume 19, 1244.

Birge, J. R., and Louveaux, F. V. 1997. *Introduction to stochastic programming*. Springer.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, 52–61.

Bonet, B., and Geffner, H. 2009. Solving pomdps: Rtdp-bel vs. point-based algorithms. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1641–1646.

Bonet, B., and Thiébaux, S. 2003. Gpt meets psr. In *13^{th} International Conference on Automated Planning and Scheduling*, 102–111.

Dolgov, D. A., and Durfee, E. H. 2005. Stationary deterministic policies for constrained mdps with multiple rewards, costs, and discount factors. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 1326–1331.

Feinberg, E., and Shwarz, A. 1995. Constrained discounted dynamic programming. *Math. of Operations Research* 21:922–945.

Hou, P.; Yeoh, W.; and Varakantham, P. 2014. Revisiting risk-sensitive mdps: New algorithms and results. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Isom, J. D.; Meyn, S. P.; and Braatz, R. D. 2008. Piecewise linear dynamic programming for constrained pomdps. In *Proceedings 23rd AAAI Conference on Artificial Intelligence*, 291–296.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1):99–134.

Kim, D.; Lee, J.; Kim, K.; and Poupart, P. 2011. Point-based value iteration for constrained pomdps. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 1968–1974.

Nilsson, N. J. 1982. *Principles of artificial intelligence*. Springer.

Ono, M., and Williams, B. C. 2008. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 3427–3432. IEEE.

Ono, M.; Kuwata, Y.; and Balaram, J. 2012. Joint chance-constrained dynamic programming. In *CDC*, 1915–1922.

Poupart, P.; Malhotra, A.; Pei, P.; Kim, K.-E.; Goh, B.; and Bowling, M. 2015. Approximate linear programming for constrained partially observable markov decision processes. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, 2164–2172.

Smallwood, R., and Sondik, E. 1973. The optimal control of partially observable markov decision processes over a finite horizon. *Operations Research* 21(5):107188.

Szer, D.; Charpillet, F.; and Zilberstein, S. 2005. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 576–583.

Teichteil-Königsbuch, F. 2012. Path-Constrained Markov Decision Processes: bridging the gap between probabilistic model-checking and decision-theoretic planning. In *ECAI*, 744–749.

Thiébaux, S., and Cordier, M.-O. 2001. Supply restoration in power distribution systems — a benchmark for planning under uncertainty. In *Proc. 6th European Conference on Planning (ECP)*, 85–95.

Undurti, A., and How, J. P. 2010. An online algorithm for constrained pomdps. In *IEEE International Conference on Robotics and Automation*, 3966–3973.

Washington, R. 1996. Incremental markov-model planning. In *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, 41–47. IEEE.

Bibliography

[1] P. Santana and B. Williams. Chance-constrained consistency for probabilistic temporal plan networks. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, 2014.

[2] Robert T Effinger. *Risk-minimizing program execution in robotic domains*. PhD thesis, MIT, 2012.

[3] Masahiro Ono, Yoshiaki Kuwata, and J Balaram. Joint chance-constrained dynamic programming. In *CDC*, pages 1915–1922, 2012.

[4] Andrew J. Wang and Brian C. Williams. Chance-constrained scheduling via conflict-directed risk allocation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.

[5] Peng Yu, Cheng Fang, and Brian C. Williams. Resolving over-constrained probabilistic temporal problems through chance constraint relaxation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, TX, July 2015.

[6] Brian C Williams and Michel D Ingham. Model-based programming: Controlling embedded systems by reasoning about hidden state. In *Principles and Practice of Constraint Programming-CP 2002*, pages 508–524. Springer, 2002.

[7] Cheng Fang, Peng Yu, and Brian C Williams. Chance-constrained probabilistic simple temporal problems. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[8] Pedro Santana, Spencer Lane, Eric Timmons, Brian Williams, and Carlos Forster. Learning hybrid models with guarded transitions. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

[9] P. Santana and B. Williams. Dynamic execution of temporal plans with sensing actions and bounded risk. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence's Doctoral Consortium*, 2014.

[10] Steven J. Levine and Brian C. Williams. Concurrent plan recognition and execution for human-robot teams. In *ICAPS-14*, 2014.

[11] David Wang and Brian C. Williams. tburton: A divide and conquer temporal planner. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*, Austin, Texas, January 2015.

[12] P. Kim, B. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*, 2001.

[13] Brian C Williams, Michel D Ingham, Seung H Chung, and Paul H Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE*, 91(1):212–237, 2003.

[14] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1):61–95, 1991.

[15] Thierry Vidal and Malik Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *ECAI*, pages 48–54, 1996.

[16] Ioannis Tsamardinos, T. Vidal, and M.E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003.

[17] Robert Effinger, B.C. Williams, Gerard Kelly, and Michael Sheehy. Dynamic Controllability of Temporally-flexible Reactive Programs. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 09)*, 2009.

[18] K Brent Venable, Michele Volpato, Bart Peintner, and Neil Yorke-Smith. Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *Workshop on Constraint Satisfaction Techniques for Planning & Scheduling*, pages 50–59, 2010.

[19] Luke Hunsberger, Roberto Posenato, and Carlo Combi. The dynamic controllability of conditional stns with uncertainty. *arXiv preprint arXiv:1212.2005*, 2012.

[20] J. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer, 1997.

[21] Hélene Fargier, Jérôme Lang, Roger Martin-Clouaire, and Thomas Schiex. A constraint satisfaction framework for decision under uncertainty. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 167–174. Morgan Kaufmann Publishers Inc., 1995.

[22] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the National Conference on Artificial Intelligence*, pages 175–180, 1996.

[23] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[24] Esther Gelle and Mihaela Sabin. Solver framework for conditional constraint satisfaction problems. In *Proceeding of European Conference on Artificial Intelligence (ECAI-06) Workshop on Configuration*, pages 14–19, 2006.

[25] S Armagan Tarim, Suresh Manandhar, and Toby Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

[26] Brian C Williams and Robert J Ragno. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.

[27] R.D. Smallwood and E.J. Sondik. The optimal control of partially observable markov decision processes over a finite horizon. *Operations Research*, 21(5):107188, 1973.

[28] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[29] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.

[30] Aditya Undurti and Jonathan P. How. An online algorithm for constrained pomdps. In *IEEE International Conference on Robotics and Automation*, pages 3966–3973, 2010.

[31] Eitan Altman. *Constrained Markov Decision Processes*, volume 7. CRC Press, 1999.

[32] Eugene Feinberg and Adam Shwarz. Constrained discounted dynamic programming. *Math. of Operations Research*, 21:922–945, 1995.

[33] Dmitri A. Dolgov and Edmund H. Durfee. Stationary deterministic policies for constrained mdps with multiple rewards, costs, and discount factors. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1326–1331, 2005.

[34] Ping Hou, William Yeoh, and Pradeep Varakantham. Revisiting risk-sensitive mdps: New algorithms and results. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.

[35] Florent Teichteil-Königsbuch. Path-Constrained Markov Decision Processes: bridging the gap between probabilistic model-checking and decision-theoretic planning. In *ECAI*, pages 744–749, 2012.

[36] Joshua D. Isom, Sean P. Meyn, and Richard D. Braatz. Piecewise linear dynamic programming for constrained pomdps. In *Proceedings 23rd AAAI Conference on Artificial Intelligence*, pages 291–296, 2008.

[37] Dongho Kim, Jaesong Lee, Kee-Eung Kim, and Pascal Poupart. Point-based value iteration for constrained pomdps. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1968–1974, 2011.

[38] Pascal Poupart, Aarti Malhotra, Pei Pei, Kee-Eung Kim, Bongseok Goh, and Michael Bowling. Approximate linear programming for constrained partially observable markov decision processes. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

[39] Nils J Nilsson. *Principles of artificial intelligence*. Springer, 1982.

[40] Thierry Vidal. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[41] Masahiro Ono and Brian C Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 3427–3432. IEEE, 2008.

[42] Lars Blackmore, Masahiro Ono, Askar Bektassov, and Brian C Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *Robotics, IEEE Transactions on*, 26(3):502–517, 2010.

[43] Anthony G. Cohn. The challenge of qualitative spatial reasoning. *ACM Computing Surveys (CSUR)*, 27(3):323–325, 1995.

[44] Michael W Hofbaur and Brian C Williams. Mode estimation of probabilistic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 253–266. Springer, 2002.

[45] Michael W Hofbaur and Brian C Williams. Hybrid estimation of complex systems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(5):2178–2191, 2004.

1.

1. Report Type

Final Report

Primary Contact E-mail

Contact email if there is a problem with the report.

psantana@mit.edu

Primary Contact Phone Number

Contact phone number if there is a problem with the report

6175992994

Organization / Institution name

Massachusetts Institute of Technology

Grant/Contract Title

The full title of the funded effort.

Robust Coordination of Autonomous Systems through Risk-sensitive, Model-based Programming and Execution

Grant/Contract Number

AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-12-1-0348

Principal Investigator Name

The full name of the principal investigator on the grant or contract.

Brian C. Williams

Program Manager

The AFOSR Program Manager currently assigned to the award

James Lawton

Reporting Period Start Date

09/01/2012

Reporting Period End Date

08/31/2015

Abstract

Unlike their human counterparts, most autonomous systems to date are not effective at characterizing or bounding mission risk. In this project, we enabled the development of risk-sensitive autonomous systems through three main contributions: first, we introduced cRMPL, an extension of RMPL where one can specify acceptable risk levels for different mission segments through the addition of chance constraints. Second, we extended the continuous planner, used by our executive, to generate and adapt plans that maximize expected utility within the risk bounds specified by the operators. Planning is performed through novel stochastic optimization algorithms that allocate user-specified risk to actions and constraints according to the benefit received. We evaluated the generality of this risk-sensitive paradigm in simulation and hardware, for autonomous air or space vehicles and humanoid logistics support robots.

Benefits include
increased number and complexity of vehicle missions for a fixed operational cost, increased robot safety around humans;
a reduction in unacceptable mission failure or robot loss, and improved mission return within defined risk levels.

Distribution Statement

This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

sf298.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

afosr_final_report_sf298.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

Fang et al., “Chance-Constrained Probabilistic Simple Temporal Problems”

Santana & Williams,“Chance-Constrained Consistency for Probabilistic Temporal Plan Networks”

Santana et al., “Learning Hybrid Models with Guarded Transitions”

Santana & Williams, “Dynamic Execution of Temporal Plans with Sensing Actions and Bounded Risk”

Santana et al., “RAO $*$: an Algorithm for Chance-Constrained POMDP’s”

Changes in research objectives (if any):

Change in AFOSR Program Manager, if any:

Extensions granted or milestones slipped, if any:

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

Sep 29, 2015 20:08:06 Success: Email Sent to: psantana@mit.edu